EXHIBIT B

FILED UNDER SEAL

IN THE UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

GOOGLE LLC, Plaintiff v. SONOS, INC., Defendant.	CASE NO. 3:20-cv-06754-WHA Related to CASE NO. 3:21-cv-07559-WHA

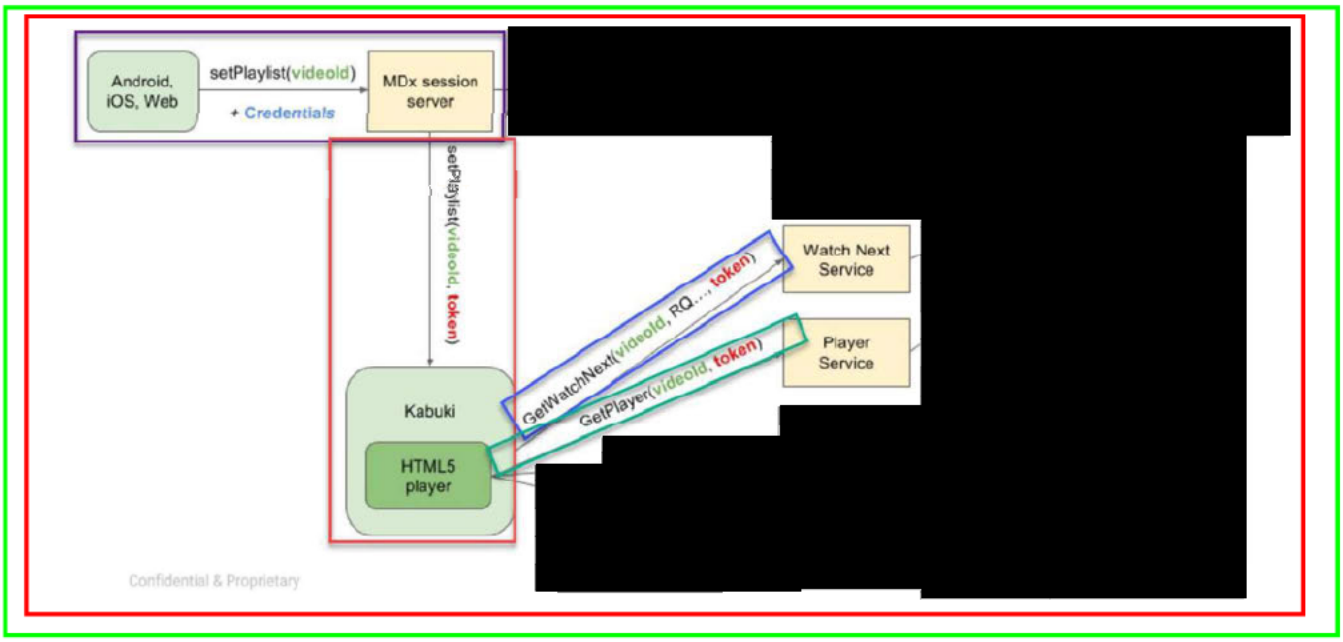
REBUTTAL EXPERT REPORT OF SAMRAT BHATTACHARJEE REGARDING NON-INFRINGEMENT OF U.S. PATENT NO. 10,779,033 AND OTHER ISSUES

HIGHLY CONFIDENTIAL AEO AND SOURCE CODE MATERIALS

(ii) Casting A User Playlist To A Cast-Receiver

80. When a user initiates a Cast session to send playback from a mobile device to a playback device, the mobile device causes the MDx server to add media items to a Shared Queue (also referred to as a "Remote Queue") through the transmission of a setPlaylist message from the mobile device to the MDx server.

81. I will now describe some of the steps that are taken when media is transferred and played back by a receiver device. A user may begin playing a media item on their mobile device (represented by the "Android, iOS, Web" box above), as I discussed above. *See* Section IX.A.1. A user may then use the Cast icon to select a receiver device(s) (represented by the "Kabuki" box) to send playback to. In this example I will assume a single playback device was selected for transfer. I will also refer to the colored boxes I have added to the image below:



GOOG-SONOSWDTX-00039491.

82. As shown in the purple box in the image above (¶81), transferring playback causes the YouTube application to send a "setPlaylist" message to the MDx server. The contents of the setPlaylist message differs across the YouTube applications. For instance, with the YouTube Main

application the setPlaylist message may include a playlist identifier (referred to as a "listID," "playlistID") that causes the MDx server to expand the playlist into a list of videoIds that are stored in the Shared Queue (also called an "RQ" or "Remote Queue") on the MDx server. *See* GOOG-SONOSWDTX-00039491 (*see* "How a Playlist Gets Added to the Shared Queue."); *see also* Nicholson Tr. at 61:11-18. ("So when we initiate a cast session, one way of doing that is to provide the MDx session server, which you see in this diagram, with a playlist ID. And then the MDx session server would be responsible for expanding that playlist into the list of videos that are in that playlist. So that's one way of initiating a session with MDx. You have a playlist ID and MDx expands that into a list of videos."). The YouTube Music application, on the other hand, expands playlists on the mobile device (referred to as "client-side expansion") and sends a setPlaylist message containing a list of videoIDs in the user's local queue to the MDx server, which are then added to the Shared Queue. *See* GOOG-SONOSWDTX-00041617 (YouTube Music Playback History in MDx Proposal) at -620; *see also* Nicholson Tr. at 61:11-62:13.

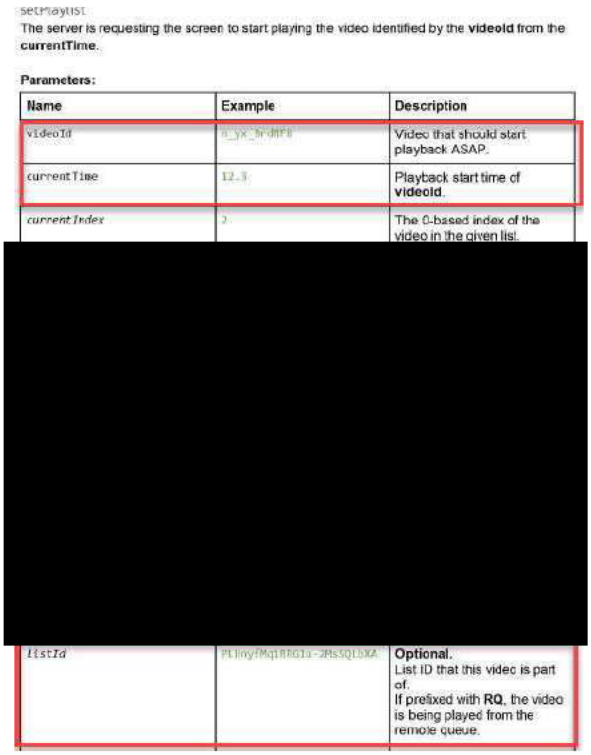
83. As shown in the red box in the image above (¶81), after receiving the setPlaylist message from the YouTube application and storing a list of videoIds in the Shared Queue, the MDx server generates and sends another setPlaylist message to the playback device. For example, the method "handleSetPlaylist" is invoked after a setPlaylist message is received by the MDx server from the YouTube application. *See* RealLoungeSessionManager.java[4], lines 1398-150. The MDx server will thereafter generate and send a new setPlaylist message from the MDx server to the playback device. *Id.* at 1496 (method sendPlaylistToScreen is called), 1566-1588 (sendPlaylistToScreen method).

[4] In 2021-02-01_YTServerMDx09292020/google3/java/com/google/youtube/lounge/browserchannel/

84. While the setPlaylist message sent from the mobile device to the MDx server and the setPlaylist message sent from the MDx server to the playback device have the same name, they contain different information. The new setPlaylist message that is created by the MDx server and sent to the playback devices is populated with new parameters. *Id.*; ParamConverter.java[5], 272-341. For example, the MDx Communication Protocol v3 specification cited in Sonos's contentions (reproduced to the right) shows some of the parameters that are included in the setPlaylist message that is sent from the MDx server to the playback devices. Among other things, the setPlaylist message sent from the MDx server to the playback devices includes an identifier for the media item that should begin playing on the playback device (current videoId), the playback start time for the media item (currentTime), and the listID (which in the case of Casting is prefixed with a "RQ" to indicate "the video is being played from the Remote Queue"-a.k.a the Shared Queue). *See* MDx Communication Protocol v3 at -00037251-252. I have added a red box in image above from the MDx Communication Protocol v3 specification to emphasize the videoId, current time, and listId values that are in the setPlaylist message.

[5] 2021-02-01_YTServerMDx09292020/google3/java/com/google/youtube/lounge/browserchannel

85. The playback device receives the setPlaylist message sent by the MDx server and the method handleMessage (loungeadapter.ts[6], line 905) processes the setPlaylist message (loungeadapter.ts, line 924). The receiver device sends a WatchNext request message to YouTube's InnerTube service and a GetPlayer request message to YouTube's Player service after receiving the setPlaylist message, which I will now discuss.


setPlaylist
The server is requesting the screen to start playing the video identified by the **videoId** from the **currentTime**.

Parameters:

Name	Example	Description
videoId	[redacted]	Video that should start playback ASAP.
currentTime	12.3	Playback start time of **videoId**.
currentIndex	2	The 0-based index of the video in the given list.
listId	[redacted]	**Optional.** List ID that this video is part of. If prefixed with **RQ**, the video is being played from the remote queue.

86. As shown in the blue box in the image above (¶81), the playback device sends a WatchNext request to YouTube's InnerTube service. The WatchNext request includes, among other things, the videoID of the media currently playing (innertube_watch_next_service.proto[7], line 97) and the playlistID corresponding to the Shared Queue at the MDx server (innertube_watch_next_service.proto, line 103). The WatchNext request is received by the InnerTube servers which invokes handle_get_watch_next (innertube_watch_next.py[8], line 349). The handle_get_watch_next function invokes _get_watch_next() (innertube_watch_next.py, line 404), which processes the request (innertube_watch_next.py, line 671) and returns to the playback device a WatchNext response (innertube_watch_service.proto; _content.py[9], starting at line 796).

[6] In 2021-02-01_YTReceivers09292020/google3/video/youtube/src/web/javascript/ library/mdx/screen_ts

[7] In 2021-02-02_YTServerInnerTubeWatchNext09292020/google3/video/youtube/api/ innertube/proto/

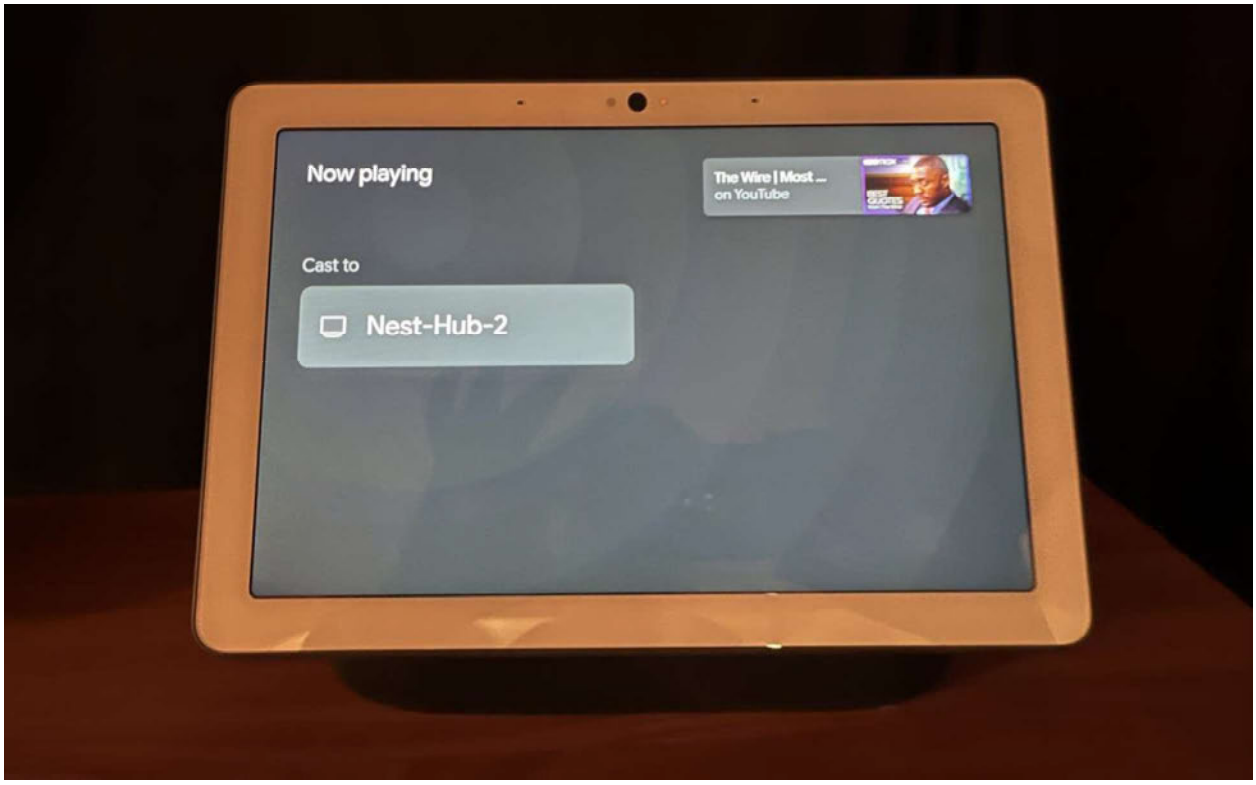
[8] In 2021-02-02_YTServerInnerTubeWatchNext09292020/google3/video/youtube/src/python/servers/innertube/watch_next/

[9] In 2021-02-02_YTServerInnerTubeWatchNext09292020/google3/video/youtube/src/python/servers/innertube/watch_next/

1. YouTube Main On A Hub Device

94. The YouTube Main application on the Hub Device includes a cast icon shown in the image below which has been annotated with a red box to emphasize the Cast icon:

95. Upon selecting the Cast icon, the YouTube Main application on the Hub Device will stop playback of whatever media item is playing locally and will then display available Cast devices on the display. An image of the screen that appears upon selecting the Cast icon is below. The YouTube Main app on the Hub Display is only able to display Cast devices that can play video content, such as another Hub Device or a Chromecast device. Schmidt Rpt., ¶181. For example in the image below the Cast speaker device on the network is not displayed. Put another way, to be able to Cast YouTube Main videos using a Hub Device, a user must own a Hub Device and at least one other Cast device capable of playing videos (e.g., a second Hub Device or a Chromecast).



96. Below are images from my testing of the Hub Devices that shows an example playback scenario. Image 1 shows a mobile device that I used to Cast playback of the video "The Wire | Most Iconic Lines" to a first Nest Hub device on the right ("Nest Hub"). Image 2 shows the Nest Hub playing back the video that is being Cast to it by the mobile device, namely the "The Wire | Most Iconic Lines." Image 3 shows a service recommended video that will play next: "Lester Freamon proves he is not a hump: Classic Wire." Image 4 shows the screen that appears after tapping the Cast icon and shows the other Nest Hubs on the network ("Nest Hub - 2"). As mentioned above, tapping the Cast icon stops playback on the Nest Hub. Image 5 shows that after I tapped the "Nest-Hub -2 icon" on the display the Nest Hub - 2 (on the left) began playback of the video. The set of service recommended videos to be played back on the Nest Hub prior to transfer (image 3) did not match the set of service recommended videos to be played back on the Nest Hub - 2 after transfer (image 6). In fact, in this example the service recommended videos was empty after transfer.

104. Google's Content Delivery Network (CDN) is sometimes referred to as "Bandaid." The CDN deploys Point of Presence ("PoP") and Edge nodes across the globe.

GOOG-SONOSNDCA-00115814 (Edge Streaming and Bandaid CDN) at slide 3.

105. [redacted] For example, Google's Life of a Video Request document explains:

106. In the YouTube system, content is broken-up into "chunks." Content is typically stored on the Bandaid servers in 2 MB chunks, which corresponds to a few seconds of a media item. GOOG-SONOSNDCA-00115893 ("Content is typically stored in 2MB chunks. Some

content, such as Live and OTF (on-the-fly transcoded) segments, are stored in smaller few-hundred KB segments, about five seconds of content each. Players make arbitrary byte-range requests, not generally aligned on 2 MB chunk boundaries."), GOOG-SONOSNDCA-00115814 (Edge Streaming and Bandaid CDN) at slide 10 ("Mostly caching video chunks: 2 MB = few seconds of video"). For instance, after the video "The Wire | Most Iconic Lines" is uploaded to the YouTube servers, it is broken-up into many chunks of video and audio and stored on the Bandaid servers. A playback device will playback the video by retrieving the chunks of video and audio content from the Bandaid servers.

107. Each YouTube song or video includes an identifier (called a videoId). A videoId by itself cannot be used to retrieve the media content for a song or video on the Bandaid servers. Instead, the process for obtaining data that can be used to retrieve the multimedia content involves a number of steps. For instance, before an alleged playback device can even make a request for the content of a song or video, it must first make a request to a "Player Service" (called a GetPlayer request). The Player Service uses a Mapping Service to decide which Bandaid server(s) the alleged playback device should talk to for a particular request and then returns to the alleged playback device Bandaid URLs that points to that Bandaid server(s). GOOG-SONOSNDCA-00115893 ("Bandaid is an HTTP-based system, and so uses URLs to request content. URLs are client-mapped, meaning that the client IP (and other parameters) are used to select a specific server hostname. Certain URL parameters are signed to mitigate abuse.").

GOOG-SONOSNDCA-00115814 (Edge Streaming and Bandaid CDN) at slide 8 ("Bandaid Mapping" will "[d]etermine, in real-time, which Bandaid node a user should talk to for a particular request (source selection) and how to route response (egress selection).").

108. For instance, the "Mapping Service" uses a videoId, itags and other information-including the viewing device's IP address, network conditions, and various other options-to generate URLs that identify Bandaid server(s) from which the receiver should request the chunks of media content. GOOG-SONOSNDCA-00115893. It then returns signed Bandaid URLs to the playback device that points to the Bandaid server(s) from which the playback device can request the chunk of the media content. The parameters of the Bandaid URLs are shown below:

GOOG-SONOSNDCA-00115814 (Edge Streaming and Bandaid CDN) at slide 9; GOOG-SONOSNDCA-00115893 (Live of a Video Request) at 3 ("To serve, ustreamer must translate a player's byte range request into the corresponding 2MB chunk(s), or players may request specific segments (Live, OTF).").[14]

X. ALLEGED ACTS OF INFRINGEMENT

A. Sonos's Allegations Of Direct Infringement Are Limited To Pixel Devices

109. I note that Dr. Schmidt only accuses the below Pixel Devices of direct infringement, which he states "are (or were) installed with one or more of the YouTube Main or YouTube Music apps." Schmidt Rpt., ¶442. Thus, I understand that for purposes of direct infringement by Google only the Pixel Devices and the YouTube Main and YouTube Music applications are at issue.

[14] Ustreamer is the name of software that runs on Bandaid servers. GOOG-SONOSNDCA-00115893 (Live of a Video Request) at 1 ("XT and EC servers run the same 'ustreamer' server software")

particular "list of media items" that he is accusing of being a "remote playback queue" or alter his source code citations, I reserve the right to respond.

162. **Second**, Limitation 1.4 recites that the "computing device is configured for playback of ***a*** remote playback queue," and Limitation 1.7 requires that a playback device take over playback responsibility of "**the** remote playback queue." Thus, for this limitation Dr. Schmidt must identify the same "remote playback queue" that he does for Limitation 1.7.

163. As I explain in further detail in connection with Limitation 1.7, Dr. Schmidt cannot do so. When Casting, each of the accused YouTube applications uses Google's MDx protocol. With MDx, a playback device plays back a Shared Queue (also known as a "Remote Queue" or "MDx queue") during a Cast session. The Shared Queue is created only *after* a Cast session is initiated. Because Limitation 1.4 is directed to playback when not Casting, it cannot involve playback of the Shared Queue. Thus, Dr. Schmidt cannot identify the same alleged "remote playback queue" for Limitations 1.4 and 1.7, as required by the claim language.

(iii) Dr. Schmidt Accuses A Local Queue-Not A Remote Queue-That Is Played Back When Not Casting

164. As I just explained, Dr. Schmidt's vague reference to a "Watch Next queue" does not identify any actual queue. In fact, to the extent Dr. Schmidt's report has identified any queue that is played back by a User Device when not Casting, it is a local, rather than a remote, queue. The express language of the claims, however, requires that the User Device be "configured for playback of a *remote* playback queue." A User Device configured for playback of a local queue does not infringe.

165. In particular, I understand Dr. Schmidt to accuse a User Device of being in a "first mode" when it is not Casting to a receiver device, and is instead playing media locally on the sender. Schmidt Rpt., ¶¶122-123. But Dr. Schmidt acknowledges that a User Device (the alleged

back "a remote playback queue" on a User Device prior to Casting, and then transferring playback of "the remote playback queue" to a receiver when Casting. The MDx playback queue that is played back on a receiver device *after* Casting is not played back on a User Device (and does not even exist) prior to Casting. Thus, the opinions I offered in connection with the Patent Showdown on the '615 patent are not inconsistent with my opinions here.

3. Hub Devices With The Accused YouTube Applications Do Not Satisfy Limitation 1.4

(A) The Accused YouTube Applications Do Not Play A Playback Queue Provided By A Third-Party Application

186. As I showed above in connection with my discussion of User Devices, the plain meaning of "remote playback queue" in view of the specification refers to a playback queue provided by a third-party application. The YouTube functionality that Sonos accuses on Hub Devices also does not play a playback queue provided by a third-party application. Thus, Dr. Schmidt cannot show that the Hub Devices play back a "remote playback queue."

(B) A Hub Device Is Not In A "Computing Device" In A "First Mode" When A Mobile Device Is Casting To It

187. It is also my opinion that a Hub Device is *not* a "computing device" operating in the claimed "first mode" when a User Device is Casting to the Hub Device.

188. Dr. Schmidt states in a single paragraph of his report that "[a] Hub Sender can begin playing back media while operating in the local playback mode based on various triggers, such as the Hub Sender detecting a voice input requesting that it start playing media from the YouTube Main or YouTube Music service or another Sender initiating a Cast session with the Hub Sender, among other possibilities." Schmidt Rpt., ¶178. However, when performing his claim analysis the only playback scenario Dr. Schmidt discusses is that of a Hub Device beginning playback when

"another Sender initiat[es] a Cast session with the Hub Sender."[18] Dr. Schmidt does not discuss the playback path for a "voice input" (or any other "trigger"), and has not shown that a Hub Device is configured to play back a remote queue, as opposed to a local queue, when playback is initiated using a voice command. Thus, my analysis below relates to the use case in which "another Sender initiat[es] a Cast session with the Hub." To the extent Dr. Schmidt is later permitted to discuss other playback scenarios, I reserve the right to respond.

189. In this use case, Dr. Schmidt opines that a Hub Device is a "computing device" in the "first mode" when a mobile device (e.g., a smartphone) running the YouTube application Casts playback to the Hub Device. This is reflected in his source code citations. In particular, Dr. Schmidt states that the "following exemplary source code demonstrates that a Hub Sender has the capability to operate in a mode in which the Hub Sender is configured for playback of the Watch Next queue," and then points to source code that is for Casting from a different mobile sender device to a Hub Device via MDx. Schmidt Rpt., ¶247 (pointing to "handleMessage" method in loungeadapter.ts and the setPlaylist message in remote.ts[19]). Although I agree that a Hub Device is playing back a cloud queue when in a Cast session with a mobile device, I disagree with Dr. Schmidt that a Hub Device is a "computing device" that is configured in the claimed "first mode" in this scenario.

190. In my opinion, a POSITA would understand a Hub Device is *not* a "computing device" in the claimed "first mode" when the mobile device is Casting to the Hub Device. The Hub Device is a Cast receiver device and is acting as a claimed "playback device" in this case.

[18] I note that Dr. Schmidt's reference to a "Cast session with the Hub Sender" is a misnomer. When another devices Casts playback to a Hub Device, the Hub Device is acting as a Hub Receiver. It is not a Hub Sender.

[19] In /2021-02-01_YTReceivers09292020/google3/video/youtube/tv/bedrock/ts/mdx/services

191. **First**, Dr. Schmidt's opinion that a Hub Device is a "computing device" operating in a "first mode" during a Cast session is in tension with the other opinions in his report. For example, for the accused User Devices (e.g., smartphone, tables and computers) Dr. Schmidt opines that the "non-Casting mode of operation" is a "first mode" and that the "Casting Mode of operation" is a "second mode":

> Google's source code confirms that each YouTube Sender is programmed to transition from the claimed "first mode" to the claimed "second mode." *See, e.g.*, Ex. 8, pp. 47-54. In this regard, the following exemplary source code for the Android version of the YouTube Main app demonstrates that a YouTube Sender has the capability to transition from the non-Casting mode to the Casting mode

Schmidt Rpt., ¶370. Thus, Dr. Schmidt has confusingly attempted to characterize the "Casting Mode of operation" as both the required "first mode" and the required "second mode."

192. **Second**, when a mobile device is Casting playback to the Hub Device the mobile device is the "computing device" configured in the "first mode" and the Hub Device is a "playback device" that begins playing after the computing device has transitioned from the "first mode" to the "second mode." In particular, the claim language recites that a "computing device" transfers "responsibility for playback" to "one or more playback devices in a media playback system" that are "available to accept playback responsibility for the remote playback queue" while in the "first mode," and then transitions to a "second mode" where it controls playback on the "playback device." The scenario Dr. Schmidt accuses begins with a user tapping the Cast icon on a computing device (e.g., a smartphone) to display Cast receivers (playback devices) that include a Hub Device. The user then selects the Hub Device (a playback device) to transfer responsibility for playback from the computing device (e.g., a smartphone) to the playback device (the Hub Device). Thus, in this use case a Hub Device is a Cast receiver that plays back media when a

computing device (e.g., a mobile device) transfers playback responsibility to the Hub Device (a playback device) and the system has transitioned out of the "first mode."

193. My opinion is supported by the specification of the '033 patent. The specification does not convey to a person of skill in the art that the inventors were in possession of a device that operated as both a "computing device" for controlling transfer of playback and a "playback device" for receiving playback responsibility at the time of filing. Indeed, such the specification doesn't contemplate that scenario. For example, the '033 patent discloses a computing device, such as Sonos controller or third-party application on a mobile device, that can include a screen for displaying an indication of one or more playback devices for selection and that the control device may receive user input for transferring playback responsibility to a playback device. '033 patent at Fig. 3 (illustrating Sonos controller device with screen), Fig. 5 (illustrating components of a controller device, including "screen" and "input interface"), 12:41-64 (describing scenario in which a third-party application is used to transfer playback to a playback device). The '033 patent separately discloses "playback devices," such as a Sonos zone player, that play back media items but that do not display or receive user input for transferring playback responsibility to another playback device. '033 patent at Figs. 2A-2C (example playback devices), Fig. 4 (showing components of a zone player, which includes no screen or input interface). In short, a person of skill in the art would understand that the '033 patent provides no written description support for Dr. Schmidt's accused use case, or a device that is both the claimed "computing device" and the claimed "playback device."

4. [1.5] while operating in the first mode, displaying a representation of one or more playback devices in a media playback system that are each i) communicatively coupled to the computing device over a data network and ii) available to accept playback responsibility for the remote playback queue;

5. [1.6] while displaying the representation of the one or more playback devices, receiving user input indicating a selection of at least one given playback device from the one or more playback devices

194. In my opinion, Dr. Schmidt has failed to show that a User Device or Hub Device satisfies Limitations 1.5 and 1.6 at least because Dr. Schmidt has failed to show that the accused YouTube applications can operate in a first mode in which the computing device is configured for playback of a remote playback queue provided by a cloud-based computing system associated with a cloud-based media service. *See* Limitation 1.4.

195. The YouTube Main functionality that Sonos accuses on Hub Devices does not satisfy these limitations for the additional reason that the Hub Device does not "display[] a representation" of the alleged playback devices "while operating in the [alleged] first mode." The claims recite that in the "first mode" the "computing device is configured for playback of a remote playback queue." Thus, a Hub Device must display the claimed representation of the alleged playback device while the "computing device is configured for playback of a remote playback queue." The Hub Devices do not. In particular, as I showed in Section IX.B.1, when the Cast icon in the YouTube Main application is selected to display the alleged playback devices, the playback of the media will stop on the Hub Device. A person of skill in the art would understand that a Hub Device that is configured for playback of the alleged remote playback queue is playing back the queue. In contrast, a Hub Device that stops playback on the queue is not configured for playing back the alleged queue.

196. The understanding that a Hub Device is no longer "configured for playback" of the alleged "remote playback queue" when playback on the Hub Device has been stopped is supported by Dr. Schmidt's opinions. For instance, Dr. Schmidt opines in his report that a User Device and Hub Device are "no longer configured for playback" of the alleged "remote playback queue" when it "stops its own playback." *See, e.g.*, Schmidt Rpt., ¶170 ("as part of transitioning from the non-

Casting mode to the Casting mode, the Sender becomes no longer configured for playback of the Watch Next queue such that, if the Sender was previously playing back media from the Watch Next queue, the YouTube Sender stops its own playback"), 200 ("as part of transitioning from the local playback mode to the remote mode, the Hub Sender becomes no longer configured for playback of the Watch Next queue such that the Hub Sender stops its own playback."). Dr. Schmidt does not point to any other configuration or de-configuration steps that User Devices and Hub Devices must perform in order to no longer be configured for playback of the alleged remote playback queue. Thus, the YouTube Main functionality that Sonos accuses on Hub Devices causes the Hub Device to "no longer be configured for playback" when displaying playback devices in much the same way that Dr. Schmidt has identified.

6. [1.7] based on receiving the user input, transmitting an instruction for the at least one given playback device to take over responsibility for playback of the remote playback queue from the computing device, wherein the instruction configures the at least one given playback device to (i) communicate with the cloud-based computing system in order to obtain data identifying a next one or more media items that are in the remote playback queue, (ii) use the obtained data to retrieve at least one media item in the remote playback queue from the cloud-based media service; and (iii) play back the retrieved at least one media item;

197. In my opinion, Dr. Schmidt has failed to show that Limitation 1.7 is satisfied by the accused User Devices and Hub Devices for several reasons. Given that there are several subparts to limitation 1.7, I have addressed the different subparts below in different sections.

(A) Dr. Schmidt Has Not Shown A User Device Causes A Playback Device Takes Over Responsibility For Playback Of "The Remote Playback Queue"

198. Initially, Limitation 1.7 again requires a "remote playback queue." As I explained in connection with Limitation 1.4, a POSITA would understand that the plain meaning of "remote playback queue" in view of the specification is a playback queue provided by a third-party

application, and the accused YouTube applications do not play back a queue provided by a third-party application. Thus, Limitation 1.7 is not satisfied for at least this reason.

199. Additionally, even accepting Dr. Schmidt's interpretation of "remote playback queue," Dr. Schmidt's opinion fails because he has not shown that the accused computing devices transfer playback responsibility of "***the*** remote playback queue" that he has accused in Limitation 1.4. In particular, I understand that a claim element in a patent is given antecedent basis, and that each subsequent reference to that same claim element is referred to as "the [element]." In the instant case, Limitation 1.4 recites that the "computing device is configured for playback of ***a*** remote playback queue," and this limitation then further requires that a playback device take over responsibility for playback of "***the*** remote playback queue." To be clear, I do not dispute that the YouTube applications are able to play back a separate cloud queue when Casting. However, that does not eliminate the requirement that Dr. Schmidt point to the same "remote playback queue" for this limitation as he did for Limitations 1.4. He has not.

200. First, Dr. Schmidt has not met his burden of establishing infringement for Limitation 1.7. Dr. Schmidt does not provide any independent analysis of the "remote playback queue" requirement for this limitation. Instead, just as with Limitation 1.4, Dr. Schmidt vaguely refers to a "WatchNext queue provided by the YouTube cloud infrastructure." Schmidt Rpt., ¶298. As I explained in connection with Limitation 1.4, the term "Watch Next queue" does not identify any actual "playback queue," let alone show that the same alleged queue is being played back in Limitations 1.4 and Limitation 1.7. *See supra*, ¶¶159-163. Thus, I disagree that Dr. Schmidt has met his burden of establishing infringement for Limitation 1.7.

201. Second, I showed that for Limitation 1.4 Dr. Schmidt has at best identified a "local queue" on a User Device (the alleged "computing device"). *See supra*, ¶¶164-177. A User Device

playing a local queue cannot transmit an instruction to "take over responsibility for playback of the remote playback queue from the computing device."

202. Third, when a User Device is Casting to a playback device, the playback device plays back a "Shared Queue" (also called a "Remote Queue," or "MDx queue"). Thus, for this limitation Dr. Schmidt must show that a User Device is configured to play back a Shared Queue prior to Casting, and that after Casting the User Device transfers playback responsibility of the Shared Queue to the playback device. Dr. Schmidt has not presented any evidence that a User Device plays back a Shared Queue when not Casting. It does not. That is because the Shared Queue is established only after a User Device initiates a Cast session. Because the Shared Queue does not exist until a user Casts playback, it cannot be "the remote playback queue" a User Device is configured to play back in Limitation 1.4 which occurs prior to Casting.

203. Dr. Schmidt points to my prior opinions during the Patent Showdown regarding the YouTube application using a cloud queue when Casting. Schmidt Rpt., ¶249. But these opinions support my current opinion that Limitation 1.7 is not satisfied because playback prior to Casting and playback after Casting do *not* use the same "playback queue." In particular, in the Patent Showdown I explained that *after* a user Casts playback to a playback device a cloud queue (the Shared Queue that I just discussed) on the MDx server is used. '615 Rebuttal Report, ¶¶82, 176-177. I explained that the cloud queue is implemented by the file SharedQueue.java, and an alleged playback device is then configured to play back this Shared Queue when in a Cast session. *Id.* Here, I agree that when Casting a playback device plays a "remote playback queue" (the Shared Queue). However, the '033 patent further requires that the "remote playback queue" played during Casting is the same "remote playback queue" played by a User Device when not Casting. That

requirement is not met here because the Shared Queue that is used when Casting does not exist prior to Casting.

204. Seemingly realizing that the Shared Queue does not exist prior to Casting, Dr. Schmidt states that the "MDx session server" manages "a *copy* of the Watch Next queue (referred to internally by Google as a 'remote queue,' 'MDx queue,' or 'shared queue') for use in the Cast session." Schmidt Rpt., ¶133. I disagree. First, as I have already mentioned, Dr. Schmidt has not identified any "Watch Next queue," let alone shown that its contents are copied to a Shared Queue. In fact, I showed above that when a user is not Casting and edits a local queue, those edits are not reflected to the YouTube servers. *See supra*, ¶¶68, 70, 171. Thus the Shared Queue cannot simply be a copy of the queue that is playing prior to Casting. Second, even if there were such a thing as a "Watch Next queue" as Dr. Schmidt alleges, a "copy of the Watch Next queue" is necessarily different than the original "Watch Next queue." Thus, Dr. Schmidt has acknowledged that when Casting a playback device does not playback "the [alleged] remote playback queue."

205. Google's documents confirm that the playback queue that is played back by a User device prior to Casting is not the same playback queue that is played back by a User Device after Casting. For instance, the presentation entitled "MDx Overview - 2015," which identifies "Ramona Bobohalma" (a creator of the YouTube Remote prior art) on its cover page explains that the Shared Queue (also known as the "RQ") that is used when Casting is "only valid for the [Cast] session." GOOG-SONOSWDTX-00040119 (YouTube MDx Overview) at -136. The presentation discloses that the Shared Queue has different properties and features than queues that are used by other accused YouTube applications when not Casting. *Id*. For example, the queues that are used by the accused YouTube applications prior to Casting allow for a user to add the same song or video to the queue multiple times (i.e., they allow "duplicates"), whereas the Shared

Queue that is used when Casting does *not* support duplicate media items. *Id.* ("RQ… no duplicates"). Similarly, a more recent presentation entitled MDx Overview 20Q3 also confirms that the "Shared Queue" is "|o|nly valid for the lifetime of the MDx session" and includes "no duplicates." GOOG-SONOSWDTX-00039988 (MDx Overview 20Q3) at -988. Similarly, below is an excerpt from the document "Orbit Queue - Android Implementation" that explains that the "local queue" in the YouTube Music application that is played back when not Casting allows for duplicate media items, while the Shared Queue (the "RQ |Remote Queue| playlist") that is played back when Casting does not:

Duplicates

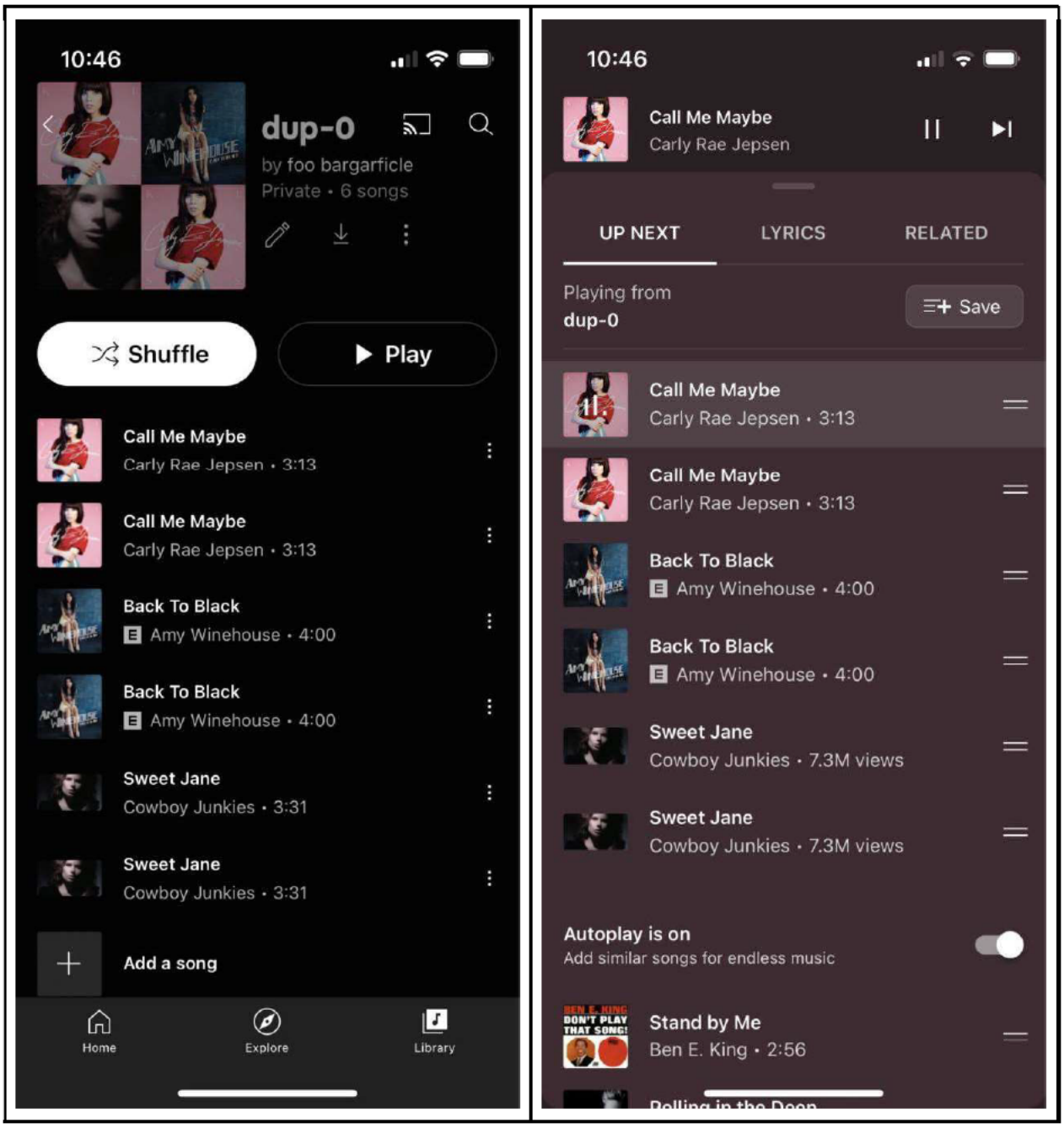
As said above. we need to manage the duplicates. This is due to the RQ playlist used on the server which currently has a limitation preventing duplicate videos to be saved in it.
Removing this limitation is currently not planned for the first implementation of the Orbit queue.

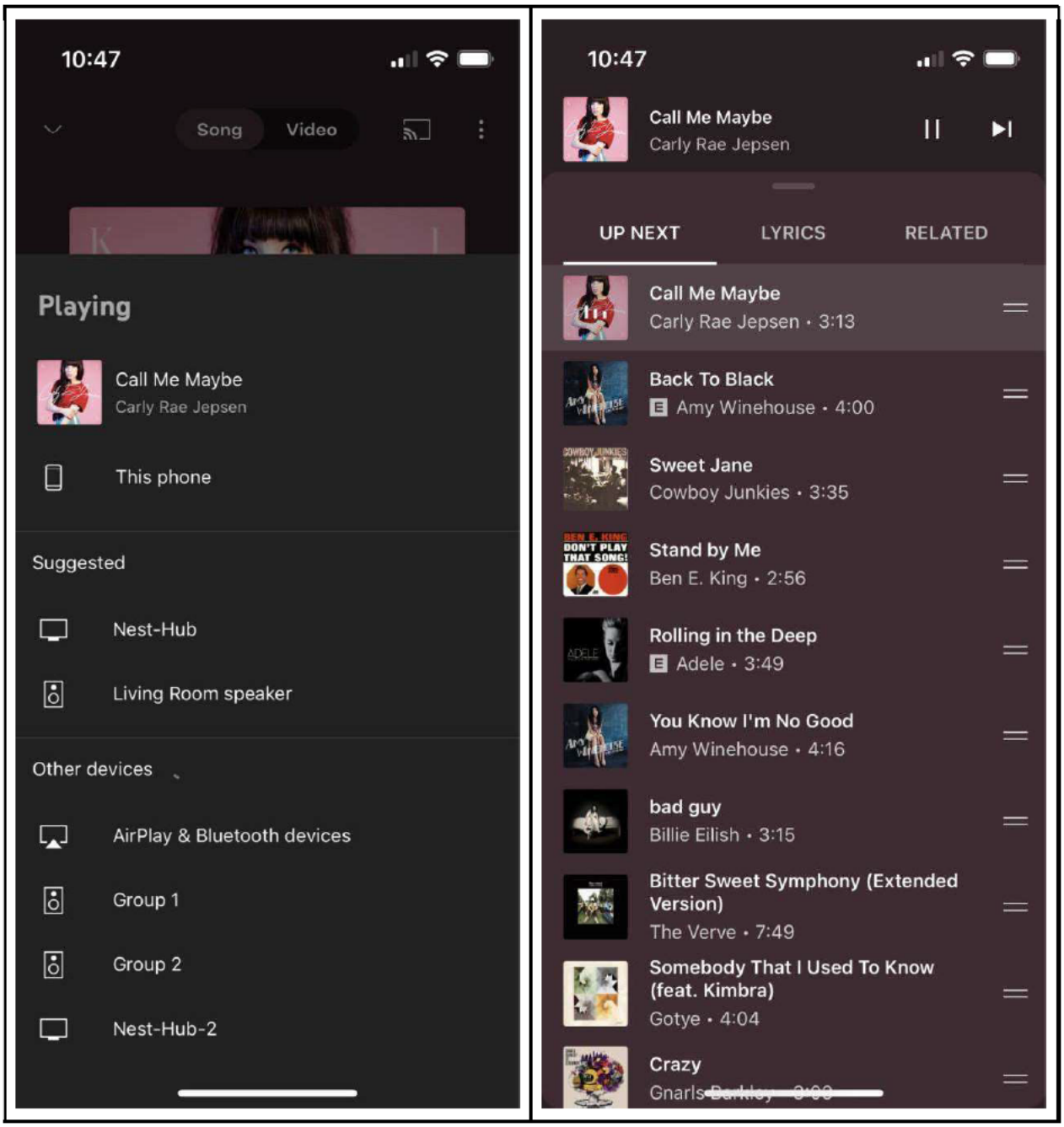
As currently the YTM local queue supports duplicates, we need to resolve these dependencies when we switch implementations.

GOOG-SONOSWDTX-00042745 (Orbit Queue - Android Implementation) at -746. The fact that the Shared Queue has different properties than the queues used by the accused YouTube applications prior to Casting further supports my opinion that the queue that the accused YouTube applications do not playback the same "remote playback queue" before and after Casting, even under Dr. Schmidt's interpretation of the term.

206. My testing of the accused YouTube applications also supports my opinion that the playback queue that is played back by a User device prior to Casting is not the same playback queue that is played back by a User Device after Casting. For instance, as I just explained, the playback queue used by the YouTube applications prior to Casting has different properties than the Shared Queue that is used when Casting. I confirmed this via my testing. For instance, I created a YouTube Music playlist containing duplicate songs (image on the top left) using my User

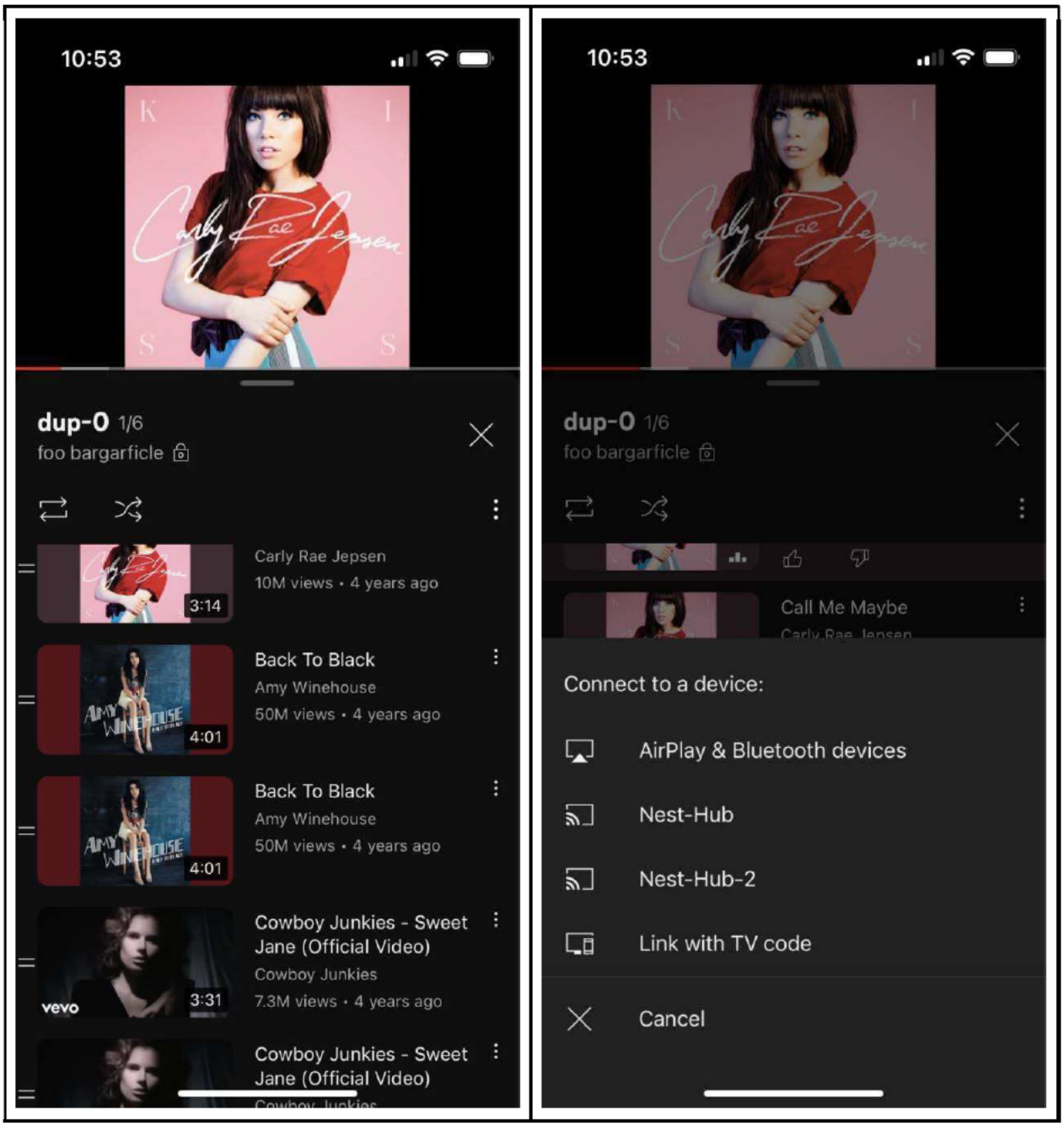
Device. While not Casting, I selected the "play" icon in order to add these items to the local queue of the YouTube Music application (image on the top right). As can be seen, the duplicate songs were added to the local queue of the YouTube Music application and began to playback on my User Device. I then tapped the Cast icon on the YouTube Music application and Cast playback to the "Living room speaker" (image on the bottom left). After Casting playback to the "Living room speaker," I observed that the duplicate media items had been eliminated from the queue (image on the bottom left). This sequence reflects the fact that when playback is cast to the Living room speaker the Living room speaker plays back a Shared Queue that is different than the local queue that was being played back on the User Device.

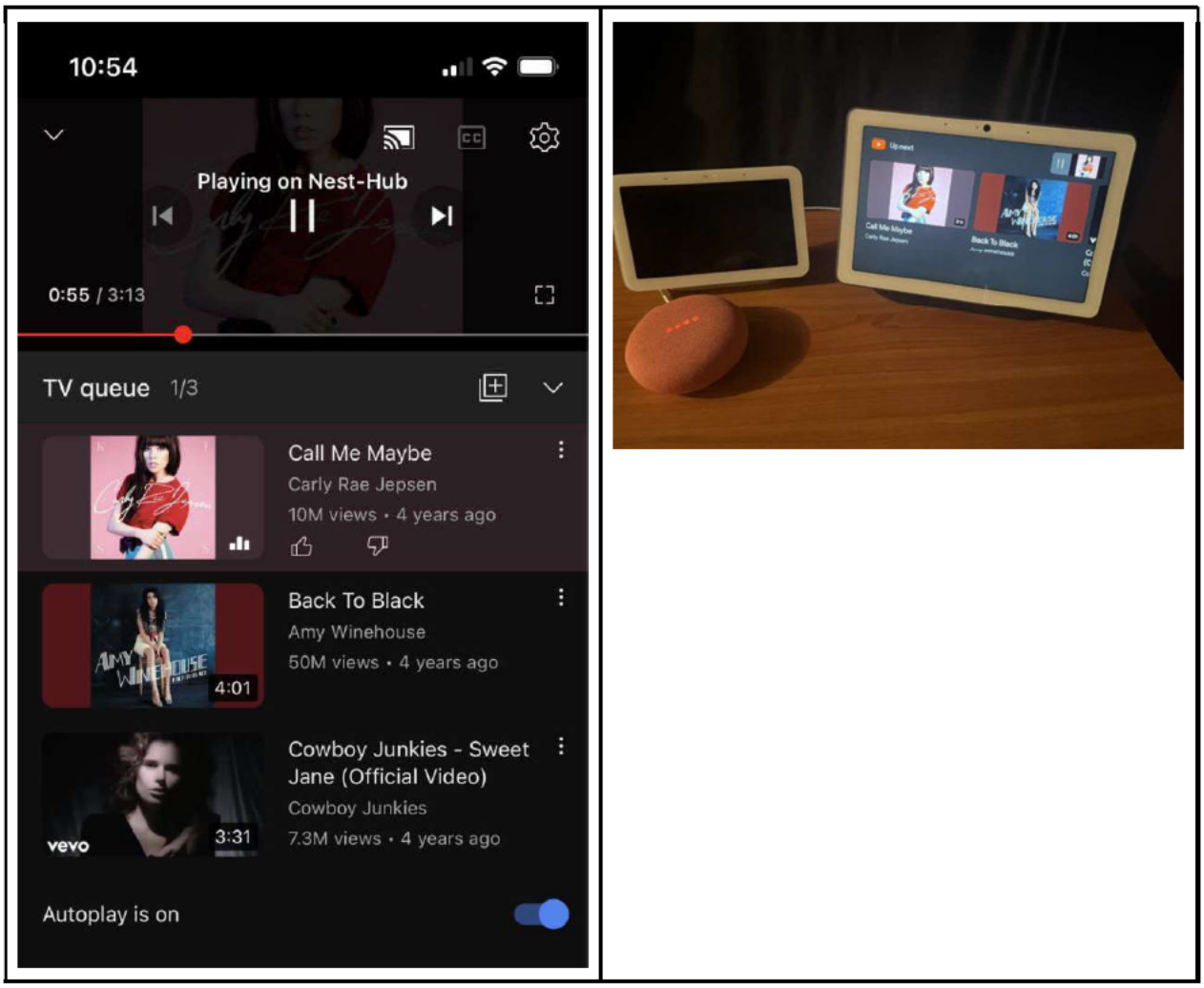

10:46
dup-0
by foo bargarficle
Private • 6 songs
Shuffle
Play
Call Me Maybe
Carly Rae Jepsen • 3:13
Call Me Maybe
Carly Rae Jepsen • 3:13
Back To Black
E Amy Winehouse • 4:00
Back To Black
E Amy Winehouse • 4:00
Sweet Jane
Cowboy Junkies • 3:31
Sweet Jane
Cowboy Junkies • 3:31
Add a song
Home
Explore
Library
10:46
Call Me Maybe
Carly Rae Jepsen
UP NEXT
LYRICS
RELATED
Playing from
dup-0
Save
Call Me Maybe
Carly Rae Jepsen • 3:13
Call Me Maybe
Carly Rae Jepsen • 3:13
Back To Black
E Amy Winehouse • 4:00
Back To Black
E Amy Winehouse • 4:00
Sweet Jane
Cowboy Junkies • 7.3M views
Sweet Jane
Cowboy Junkies • 7.3M views
Autoplay is on
Add similar songs for endless music
Stand by Me
Ben E. King • 2:56



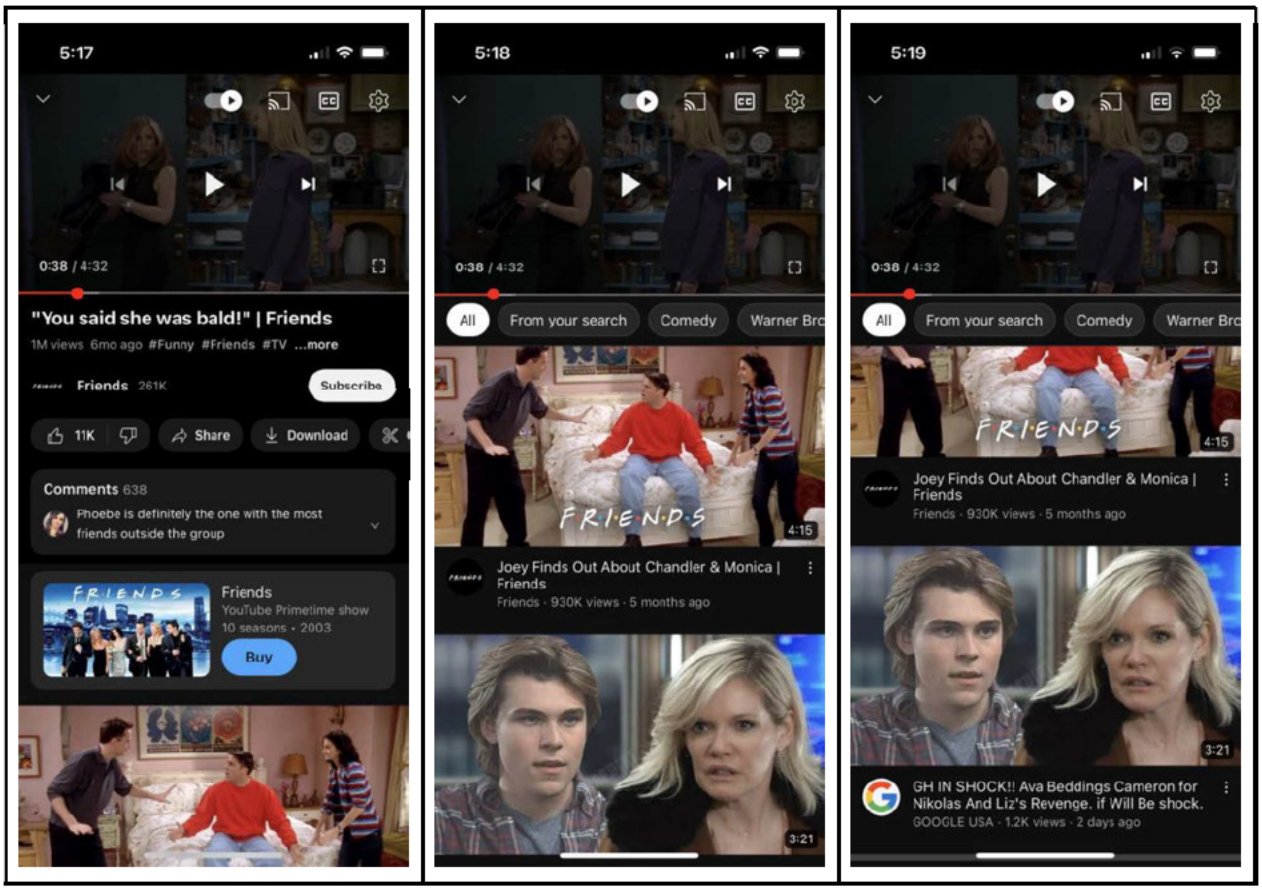
207. I also performed a similar test for the YouTube Main application. In particular, I began playback on my User Device of a queue containing duplicate media items (top left image). I then Cast playback to the Nest-Hub device on my network by tapping the Cast icon and selecting the Nest-Hub (top right image). Just as in the case of YouTube Music, after Casting playback to the Nest-Hub I observed that the duplicate media items had been eliminated from the queue

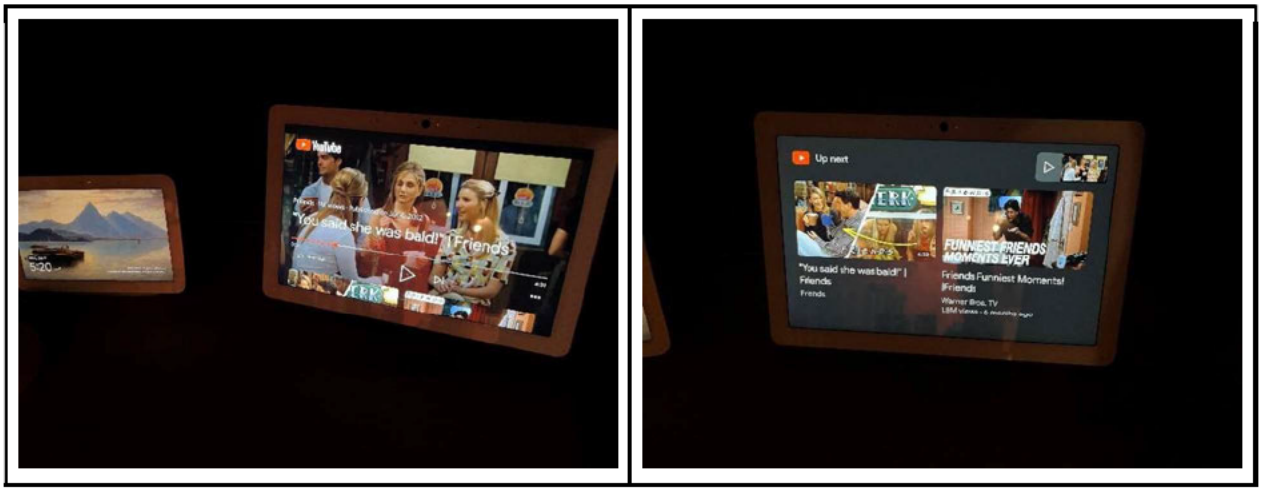
(images on the bottom left and right). Again, this sequence reflects the fact that the play back queue that is played back prior to casting is not the Shared Queue that is played back after casting.





208. As another example, my testing of the YouTube Main application shows that new, different Autoplay items are generated when a user initiates a Cast session. For instance, I began playback on a single media item while not Casting and with Autoplay enabled, namely the video "You said she was bald! | Friends" (image on the left). After Casting playback to the Nest-Hub, I observed that the Autoplay videos changed. Thus, when a User Device Casts playback to a playback device using the YouTube Main application it does not transfer playback responsibility for the *same* "playback queue."





209. As yet another example, my testing of the YouTube Kids application also shows that new, different Autoplay items are generated when a user initiates a Cast session. For instance, I showed above that after Casting playback from the YouTube Kids application to the Hub Device the list of service-recommended videos may change. *See supra*, ¶¶56-57. Thus, when a User

Device Casts playback to a playback device using the YouTube Kids application it does not transfer playback responsibility for the *same* "playback queue."

210. As yet a further example, my testing of the YouTube TV application shows that when a User Device Cast playback of an episode of a television show to a playback device, the playback device does not "take over responsibility for playback of [any alleged] remote playback queue." Rather, I showed above that when playback of an episode is Cast to a playback device, the playback device will stop playback following completion of the episode, as shown in the image below. *See also supra*, ¶¶58-59.

(i) "based on receiving the user input, transmitting an instruction for the at least one given playback device to take over responsibility for playback of the remote playback queue from the computing device, wherein the instruction configures the at least one given playback device to ..."

211. Dr. Schmidt accuses two different messages-a launch message and a setPlaylist message-and alleges that together they form the claimed instruction that configures the at least one given playback device to perform steps (i)-(iii) of this limitation. Schmidt Rpt., ¶298. I disagree with Dr. Schmidt that the launch and setPlaylist message can be the claimed instruction.

212. **First**, the launch message alone cannot be the claimed instruction because it does not configure the at least one given playback device to perform steps (i)-(iii) of this limitation. In particular, after receiving only a launch message a receiver device cannot perform the steps Dr. Schmidt accuses of satisfying steps (i)-(iii), for instance sending or receiving a WatchNextResponse. Dr. Schmidt appear to agree as he describes the "launch message" as merely "instruct[ing] the Receiver device to launch the applicable receiver-side YouTube app," and relies upon the setPlaylist message as the alleged message that "instructs the Receiver to take over playing back the media item that the user device was previously responsible for playing back… as well as upcoming media items from the WatchNext queue." Schmidt Rpt., ¶298.

213. **Second**, the setPlaylist message that Dr. Schmidt relies upon is not the claimed instruction. The claim language requires that the "computing device" send the instruction. In the accused YouTube applications the setPlaylist message sent by an accused YouTube application running on a sender device (the alleged "computing device") is not even received by the playback device-it is only received by an MDx server. Thus, the setPlaylist message sent by the YouTube application is not an instruction that configures the at least one given playback device to operate in accordance with step (i)-(iii) of this limitation.

214. Dr. Schmidt does not dispute that the setPlaylist message sent from the computing device is received by an MDx server, but opines that the claim language does not require the computing device send the claimed instruction "directly to" the "at least one given playback

device." Schmidt Rpt., ¶¶312-314. But the issue is not whether the instruction from the computing device to the playback device requires a direct or indirect transmission. Instead, the issue is that the instruction sent from the computing device must be the same instruction received by the playback device. This is mandated by the plain language of the claim, which identifies the instruction sent by the computing device as the same instruction that configures the playback device.

215. Dr. Schmidt cites to certain documents and source code that he opines shows that the setPlaylist message sent from the computing device to the MDx server is "relayed" to the alleged playback device. Schmidt Rpt., ¶¶299-300, 305, 315. But the setPlaylist message sent from the alleged computing device to the MDx server is not the same setPlaylist message that is sent from the MDx server to the alleged playback device. After receiving this setPlaylist message, the MDx server invokes the handleMessage method (RealLoungeSessionManager, line 740) and creates a new, different setPlaylist message to send to the playback device (RealLoungeSessionManager.java, line 1496). The format of the setPlaylist message that is sent from the MDx server to the playback device is described in the MDx Communication Protocol v3 specification and is different than that of the setPlaylist message sent from the User Device to the MDx server. *See* GOOG-SONOSWDTX-00037243 at 251-252. By way of example, the document below shows that the setPlaylist message sent from a User Device running the YouTube Music application to the MDx (also known as Orbit) server (which I have annotated as "setPlaylist 1") contains information that is different than the setPlaylist message sent by the MDx server to the TV (which I have annotated as "setPlaylist 2"):

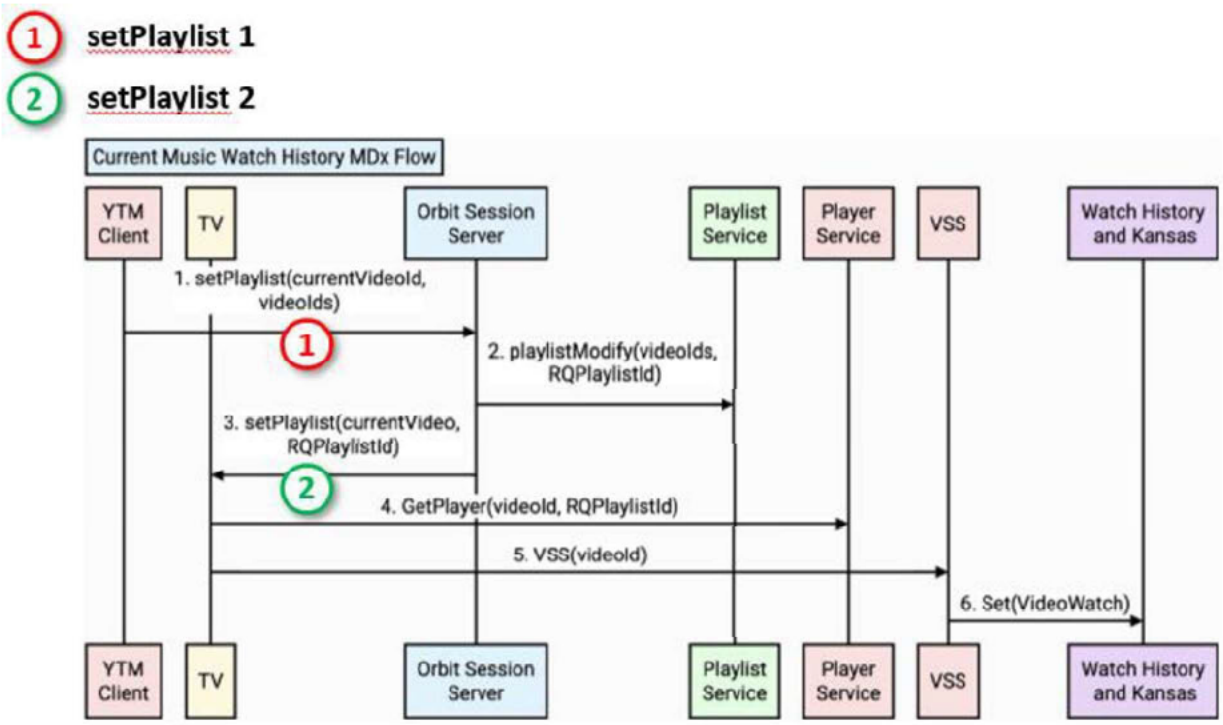


Figure 1. MDx Playback Triggered By YTM

GOOG-SONOSWDTX-00041620. For instance, the setPlaylist 1 message contain a list of videoIds in the local queue of the YouTube music application, while the setPlaylist 2 message sent from the MDx server contains information about the Shared Queue (e.g., the RQPlaylistId). Thus, while the name of the message sent from the computing device and the name of the message sent from the MDx server are the same (setPlaylist) they are different messages containing different information.

216. **Third**, I understand that the parties dispute the plain meaning of the term "an instruction for the at least one given playback device to take over responsibility for playback of the remote playback queue from the computing device, wherein the instruction configures the at least one given playback device to (i) …, (ii) …, and (iii) …." I understand that Dr. Schmidt contends that this limitation does not require a singular instruction that configures a playback device to do each of the functions recited in steps (i)-(iii), and instead may encompass multiple

servers." *Id*, ¶329. But Dr. Schmidt's analysis is conclusory and he fails to explain how the videoId is used to retrieve and play back the next media item in a "remote playback queue."

222. In my opinion, an accused playback device does not use a videoId "to retrieve at least one media item in the remote playback queue from the cloud-based media service." In the accused YouTube system, each song or video includes an identifier (called a videoId) and the same song or video may exist on many different Bandaid CDN servers at a given time. A receiver device cannot use a videoId to retrieve a media item from a Bandaid CDN. Indeed, if the receiver device sent the videoId to a Bandaid CDN, the Bandaid CDN would not return any media item.

223. Instead, a significantly more complex process occurs: in the accused YouTube systems media items are divided into "chunks" that correspond to a few seconds of audio or video. A single media item may be broken into many different chunks of audio and video content. To obtain the media item, the playback device makes a GetPlayer request to obtain signed Bandaid URLs from the servers of the "Player Service." The Bandaid URLs points to a particular, separate Bandaid CDN server(s) to use to obtain the media item (or, more specifically, the chunks of the media item). The signed Bandaid URLs are thus the data that the playback device uses to request the media item from the Bandaid CDN servers. The fact that the YouTube system does not use a videoId to retrieve media content—and instead uses a signed Bandaid URL—is a deliberate design required for security and performance of the YouTube system. For instance, a signed Bandaid URL expires after a certain amount of time. GOOG-SONOSWDTX-00052129 at -133 ("The URLs expire after a certain time and do not work after expiry.") By requiring that signed Bandaid URLs (and not a videoId) be the data that is used to request the media item, the YouTube system provides security in the event a malicious actor were to intercept the Bandaid URLs. The Bandaid URL would expire and thus limit the ability of the malicious actor to use it to obtain media he or

she is unauthorized to access. Just as importantly, the Player Service generates Bandaid URLs at the time of playback taking into account various factors. For instance, the videoId is input to a "Mapping Service" that uses information other than the videoId-including the viewing device's IP address, network conditions, and various other options to generate Bandiad URLs that identify a Bandaid server from which the receiver should request the content. Notably, the Mapping Service uses factors, such as the client IP address and network conditions, that are entirely unrelated to the videoId, which reinforces my view that the videoId is not "translated" into the URL. And because the Mapping Service uses dynamic factors, the same videoId can return different URLs for each user or for the same user each time the video is played back. Due to the sheer scale of the YouTube system these constraints have to be considered in order to serve media in scalable manner. The design of this YouTube system in which a videoId is used, along with other information, to retrieve ephemeral Bandaid URLs (rather than the media content) has specific security and network properties that are necessary to provide content delivery at the large scale YouTube provides.

224. My opinion is also supported by the disclosures in the '033 patent. Like the asserted claims, the '033 patent describes a simplistic two step process for retrieving media content. First, a playback device receives a URI (e.g., a UR) for a media item in a queue provided by a third-party application ("obtain data identifying a next one or more media items that are in the remote playback queue"). Then the playback device passes that URI to a cloud-based media service and "simply receive[s] content." '033 patent at 12:41-64; *see also id.* at 12:35-40 ("playback device 764, 774 can provide a song identifier, song name, playlist identifier, playlist name, genre, preference, and so on, and/or simply receive content from a connected system via the cloud."). Thus, the '033 patent contemplates a simplistic two-step process (obtain URI and use URI to fetch content), whereas the YouTube system is substantially more complex.

225. Because Dr. Schmidt has not shown that the alleged "instruction" (launch and setPlaylist messages) configures the playback device to communicate with the cloud-based computing system in order to obtain a Bandaid URL for a next one or more media items that are in the remote playback queue, Dr. Schmidt has failed to show this limitation is satisfied.

(A) Hub Devices Do Not Satisfy Limitation 1.7

(i) Dr. Schmidt Has Not Identified A "Remote Playback Queue"

226. As I previously explained in connection with Limitation 1.4, the YouTube functionality that Sonos accuses on Hub Devices also does not play a playback queue provided by a third-party application. Thus, Dr. Schmidt cannot show infringement at least because the Hub Devices do not use a "remote playback queue."

(ii) "based on receiving the user input, transmitting an instruction for the at least one given playback device to take over responsibility for playback of the remote playback queue from the computing device, wherein the instruction configures the at least one given playback device to…"

227. Dr. Schmidt accuses two different messages—a launch message and a ResumeSession message—and alleges that together they form the claimed instruction that configures the at least one given playback device to perform steps (i)-(iii) of this limitation. Schmidt Rpt., ¶307. Initially, as I showed above this limitation requires a singular instruction that configures a playback device to do each of the functions recited in steps (i)-(iii). *See supra* ¶¶216-219. Because Dr. Schmidt relies upon multiple different instructions that he alleges collectively configure a playback device to perform steps (i)-(iii), he has not shown this limitation is satisfied.

228. Additionally, the claim language requires that the instruction be sent by the "computing device." Here, the "launch message" and "ResumeSession message" are sent by a Hub Device, which is the claimed "playback device," not the claimed "computing device," in the

scenario Dr. Schmidt accuses of infringement. Indeed, as I explained in connection with Limitation 1.4, for Hub Devices Dr. Schmidt accuses the scenario in which a mobile device (e.g., a smartphone) Casts playback to the Hub Device. *See supra* ¶189. The mobile device (which is the one transferring playback responsibility to the Hub Device) is the "computing device" in this scenario, and the Hub Device which is accepting playback responsibility from the mobile device is the "playback device." Thus, Dr. Schmidt was required to identify an instruction sent by the mobile device, which he has failed to do.

(iii) *"wherein the instruction configures the at least one given playback device to (i) communicate with the cloud-based computing system in order to obtain data identifying a next one or more media items that are in the remote playback queue, (ii) use the obtained data to retrieve at least one media item in the remote playback queue from the cloud-based media service; and (iii) play back the retrieved at least one media item"*

229. For this portion of Limitation 1.7, Dr. Schmidt's accusations against the Hub Device are the same as those he provided for a User Device. Thus, in my opinion a Hub Device does not satisfy this portion of the limitation for the same reason I discussed above.

7. **[1.8] detecting an indication that playback responsibility for the remote playback queue has been successfully transferred from the computing device to the at least one given playback device;**

8. **[1.9] after detecting the indication, transitioning from i) the first mode in which the computing device is configured for playback of the remote playback queue to ii) a second mode in which the computing device is configured to control the at least one given playback device's playback of the remote playback queue and the computing device is no longer configured for playback of the remote playback queue.**

230. In my opinion, Dr. Schmidt has failed to show that Limitations 1.8 and 1.9 are satisfied at least because Dr. Schmidt has failed to show that the accused YouTube applications can operate in the claimed "first mode" or that the a playback device in the system can accept playback responsibility for a "remote playback queue." *See* Limitation 1.4.

detected. Thus, Dr. Schmidt has failed to show that the alleged computing devices transition from the first mode to the second mode *after* detecting a CONNECTED event.

(B) Hub Devices With The Accused YouTube Applications Do Not Transition To The Claimed Second Mode After Detecting The Claimed Indication

235. As I showed above in connection with Limitation 1.4, Dr. Schmidt contends that a Hub Device is a "computing device" configured in the "first mode" where a User Device is Casting playback to a Hub Display. I showed why this is incorrect. *See supra*, ¶¶187-193. Even setting this issue aside, Dr. Schmidt has not shown a Hub Device satisfies Limitations 1.8 and 1.9.

236. For Limitations 1.8, Dr. Schmidt appears to allege that if a user selects another destination device for playback (e.g., a second Hub Device), the first Hub Display (that was previously playing the casted content) will send a ResumeSession message to the second destination device (that is newly selected to play the casted content instead) that will allegedly result in the first Hub Display receiving a response that it processes with the OnTransferComplete method. Schmidt Rpt., ¶¶349, 356. Dr. Schmidt appears to accuse the response to the ResumeSession message of being the claimed "indication."

237. For Limitation 1.9, the claim language sets forth a sequence by which the Hub Device must transition from the "first mode" in which it is "configured for playback" to "a second mode" where it is "no longer configured for playback" only after it detects the claimed "indication." Dr. Schmidt fails to show that the accused Hub Devices perform the claim requirements in the correct sequence.

238. In his analysis of Limitation 1.9, Dr. Schmidt merely states that "[t]he following exemplary source code demonstrates that a Hub Sender has the capability to transition from local playback mode to the remote mode" and cites to the following passages of code:

configured for playback of the remote playback queue" *after* (as opposed to prior to or simultaneously with) detecting the alleged indication.

239. The testing I performed also illustrates that the Hub Device is no longer in the alleged "first mode" at the time the alleged indication is detected.

240. For instance, I showed in connection with Limitation 1.5 that the YouTube Main application on the Hub Device will stop playback at the time the user selects the Cast icon to display the device-picker. Put another way, playback on the Hub Device is stopped before a user selects a device to Cast to, and before the Hub Device sends a ResumeSession message. Because the accused Hub Device stops playback *before* (not *after*) receiving the alleged indication that Dr. Schmidt points to (a response to the ResumeSession message), Dr. Schmidt cannot show the Hub Device is no longer configured for playback of the alleged remote playback queue "after detecting the indication," as required by the claims.

241. I also showed above that when a user Casts playback to a second Hub Device the YouTube Music application on the Hub Device continues playback after the alleged "indication" is detected, unless a user chooses to stop playback on the Hub Device. *See supra*, ¶¶97-102. Thus, a Hub Device with YouTube Music also does not infringe because after detecting the alleged indication the Hub Device does not transition to "a second mode in which" it is no longer configured for playback of the alleged remote playback queue."

B. Claim 2

1. "The computing device of claim 1, wherein the instruction comprises an instruction for the cloud-based computing system associated with the media service to provide the data identifying the next one or more media items to the given playback device for use in retrieving the at least one media item from the cloud-based computing system associated with the cloud-based media service."

242. Dr. Schmidt does not accuse Hub Devices of infringing Claim 2. Schmidt Rpt., ¶120. I agree with Dr. Schmidt that a Hub device does not infringe Claim 2.

243. Dr. Schmidt accuses only User Devices provisioned with the accused YouTube applications of infringing Claim 2. I understand that Claim 2 depends on Claim 1. I further understand that a dependent claim incorporates all of the limitations from the independent claim. Thus, for all of the reasons set forth above in my discussion of Claim 1, it is my opinion that Dr. Schmidt has failed to demonstrate that this portion of the claim is satisfied.

244. Dr. Schmidt has also failed to show Claim 2 is satisfied for additional reasons. For Claim 1, Dr. Schmidt identified a launch message and setPlaylist message as "the instruction." For the additional limitation of Claim 2, Dr. Schmidt further opines that the "setPlaylist" message also comprises "an instruction for the cloud-based computing system associated with the media service to provide the data identifying the next one or more media items to the given playback device for use in retrieving the at least one media item from the cloud-based computing system associated with the cloud-based media service." Schmidt Rpt., ¶¶380-381. In my opinion the setPlaylist message that Dr. Schmidt points to does not satisfy Claim 2.

245. The setPlaylist message sent from a sender device is not "an instruction for the cloud-based computing system associated with the media service to provide data identifying the next one or more media items to the given playback device." In particular, the setPlaylist message sent from a sender device is received by an MDx server in the cloud. Upon receiving the setPlaylist message from the sender device, the MDx server does not "provide data identifying the next one or more media items to the given playback device." Instead, the MDx server creates a new setPlaylist message that contains different information that the setPlaylist message sent from the computing device. The new setPlaylist message sent from the MDx server includes, among other

things, a videoId which identifies the *current* media item that should be played back on the playback device-it does not include a videoId for the "next one or more media items" that should be played. And, of course, because the videoId sent from the MDx server to the playback device in a setPlaylist message does not identify the "next one or more media items," it also is not "for use in retrieving" the next one or more media items.

246. Dr. Schmidt appears to agree that the setPlaylist message sent by the User Device to the MDx server does not cause the MDx server to send the playback device a "next one or more media items." Instead, Dr. Schmidt opines that the setPlaylist message sent by the User Device "to a Receiver via the MDx server" causes the receiver to send a WatchNext request and receive in response a WatchNext response that contains "data identifying the next one or more media items." Schmidt Rpt., ¶¶380-384. I disagree. As I explained in connection with Limitation 1.7, the setPlaylist message sent from the MDx server to the receiver device is a different message with different information than the setPlaylist message sent from the sender device to the MDx server. Because the claims require the sender device (the alleged "computing device") transmit the claimed instruction, a setPlaylist message sent from an MDx server does not satisfy the claims.

247. Assuming that the setPlaylist message sent from the MDx server can be characterized as the same setPlaylist message sent from the sender device (it cannot), this limitation is still not met. The plain language of the claim requires that "the instruction" (which Dr. Schmidt identifies as the setPlaylist sent from the MDx server to the playback device) "comprises an instruction for the cloud-based computing system associated with the media service to provide the data identifying the next one or more media items to the given playback device." Dr. Schmidt opines that the setPlaylist message satisfies the additional limitation of Claim 2 because it "indirectly" causes "the YouTube cloud infrastructure (and specifically, the WatchNext

capable of video playback is needed. In addition, the accused YouTube Main functionality on a Hub Device also does not permit Casting to a group of devices.

XIV. RESPONSE TO DR. SCHMIDT'S NON-INFRINGING ALTERNATIVE OPINIONS

A. Non-Infringing Alternative - Continue Playback On The Computing Device After Transferring Playback

278. All of the asserted claims require "transitioning from i) the first mode in which the computing device is configured for playback of the remote playback queue to ii) a second mode in which the computing device is configured to control the at least one given playback device's playback of the remote playback queue and the computing device is no longer configured for playback of the remote playback queue." As I showed above, the accused YouTube applications do not infringe this limitation. Nevertheless, even if Dr. Schmidt's infringement opinions were accepted, as I showed in my opening report, a non-infringing alternative that was available to Google at the time the '033 patent issued was a "set and forget" operation. The set and forget operation would modify the accused YouTube applications such that the accused YouTube applications would not satisfy the portion of this limitation that requires transitioning to a second mode in which "the computing device is no longer configured for playback of the remote playback queue," even accepting Dr. Schmidt's interpretation of this claim limitation.

279. Initially, Dr. Schmidt does not dispute that set and forget operation would not infringe the '033 patent. I agree with Dr. Schmidt that this alternative is non-infringing.

280. Dr. Schmidt wrongly suggests that this non-infringing alternative would apply only to the YouTube Main app. Schmidt Rpt., ¶514. More specifically, Dr. Schmidt points to a statement from Google that "playback can be continued with only the video alone (and the audio muted)" and states that this would "not account for YouTube Music's music-only playback." Schmidt Rpt., ¶514. Dr. Schmidt's position appears to be inconsistent with his infringement theory

for Limitations 1.4 and Limitation 1.9. For Limitations 1.4 and 1.9, Dr. Schmidt opines that an accused YouTube application is still "configured for playback of the remote playback queue" even when it is paused and not playing back the media. *See, e.g., supra* ¶¶232-234 (Limitations 1.8-1.9). While I do not agree with Dr. Schmidt, applying his interpretation would mean that music-only playback of the remote playback queue would be non-infringing (even where the audio was muted). Moreover, the alternative could be implemented so that all of the YouTube applications default to pausing playback of the media (whether audio or video) on the computing device upon transfer. In either implementation, each of the accused YouTube applications would not infringe under Dr. Schmidt's interpretation because they would not satisfy the step of transitioning to a second mode in which "the computing device is no longer configured for playback of the remote playback queue." Even setting aside Dr. Schmidt's interpretation, this non-infringing alternative could also be implemented so that the video and audio would continue to playback on the mobile device after transferring playback, giving the user the option to continue the playback on the mobile device or pause it at a later time. Thus, this non-infringing alternative was available to Google as of the date the '033 patent issued for each of the YouTube Main, YouTube Music, YouTube Kids, and YouTube TV applications.

281. Dr. Schmidt also opines that the non-infringing alternative would not have been commercially acceptable for YouTube Music's music-only playback because users allegedly do not want audio being output from their phone and a Cast device at the same time. Schmidt Rpt., ¶520. But Dr. Schmidt has not shown that his statement applies to all users. Users may have different preferences, and, as I just explained, this alternative would provide users the option to continue playback on the phone or to not continue playback on the phone, according to their preference. Indeed, the alternative could be implemented such that audio is muted or paused on

the user's phone upon transfer. Users would then have the option to resume audio output on their phone or to not resume audio output on their phone. Alternatively, the default behavior upon transferring playback could be to continue playback, and users would then have the option to pause playback if they so desire. Thus, this option would provide users with additional features and functionality.

282. Dr. Schmidt further asserts that this alternative is not technically feasible. Schmidt Rpt., ¶¶516-518. I disagree.

283. First, Dr. Schmidt alleges that YouTube Music has "streaming limitations that prevent streaming media from more than one device at a time." Schmidt Rpt., ¶517. Initially, Dr. Schmidt's assertion is limited to the YouTube Music application, and Dr. Schmidt does not argue that any streaming limitations would prevent Google from implementing this alternative for the other accused YouTube applications. Moreover, I disagree that streaming limitations would prevent Google from implementing this alternative for the YouTube Music application. As I explained earlier, for purposes of infringement Dr. Schmidt has opined that the limitation requiring that the alleged computing device "is no longer configured for playback of the [alleged] remote playback queue" is not satisfied even where a computing device's playback has been stopped. *See* ¶¶232-234. Put another way, Dr. Schmidt contends that an alleged computing device remains configured for playback of the alleged remote playback queue when it is paused and able to resume playback of the alleged remote playback queue at a later time. *Id.* Accordingly, even if, as Dr. Schmidt asserts, streaming limitations restrict playback to a single device at a given time, this alternative would not be impacted because the alleged computing device could simply pause its playback upon transferring playback to the Cast device such that only one device would play back at a given time. Further, YouTube Music offers a free subscription whereby users are able to play

back media on multiple devices, as well as premium subscriptions that permit multiple devices to playback media at the same time. Thus, if a user chooses to resume playback on the computing device during a Cast session, the alleged computing device could play back media. Users would thus be provided with additional functionality in this alternative because they could play back media on their Cast receiver device and also have the option to play back media on the alleged computing device.

284. Second, Dr. Schmidt alleges that "assuming Google could have resolved the aforementioned streaming limitations, it would have been hard to enable simultaneous playback of a given media item at both the Sender and Receiver without introducing new technical challenges." Schmidt Rpt., ¶518. But Dr. Schmidt fails to explain why simultaneous playback would be necessary to implement this non-infringing alternative. As already mentioned, this alternative could simply pause the playback on the alleged computing device. A user would then have the option to un-pause the playback on the alleged computing device, as the user desires. For instance, a user may be Casting a song or video to a receiver device in their living room, and then move to his or her bedroom. The user could then play back media on the alleged computing device, while other family members continue to watch the media on their living room device. In short, this alternative provides users with additional features that they can choose to utilize according to their preferences.

285. Third, Dr. Schmidt also opines that this alternative "would have presented technical challenges associated with how to control the Receiver's playback of the given media item since the Sender's transport control would ostensibly control its own playback of the given media item." Schmidt Rpt., ¶519. I disagree. A user could control the receiver's playback of the given media item using the transport controls in the Google Home application, which is the application the user

already uses to set-up and manage receiver devices that support Casting. In fact, this alternative could be implemented such that transferring playback to a receiver device automatically opens up the Google Home application and displays the transport controls for controlling the receiver's playback of the given media item, including transport controls for pausing, forwarding, and rewinding media playback, and skipping to the next or previous media items in the alleged remote playback queue. If the user chooses to utilize the additional functionality provided by this alternative (namely, the ability to resume playback on the alleged playback device), the user could simply switch back to the YouTube application and use the transport controls in the YouTube application. Further, I note that the ability to control multiple devices in this manner is already deployed within the Google Home application.

286. Fourth, Dr. Schmidt asserts that this alternative would "drain[] a Sender's battery." Schmidt Rpt., ¶520. Dr. Schmidt's assertion is conclusory and he provides no explanation for why this alternative would result in additional battery drain. Dr. Schmidt has not demonstrated that this alleged battery drain (if any) would be of material concern to users. Indeed, the documents Dr. Schmidt cites for his assertion merely state that a user can reduce the drain on their phone's battery by playing back media on the receiver device (e.g., a Chromecast) instead of the user's phone. Schdmit Rpt., ¶471 (citing documents). But in the non-infringing alternative proposed here, playback on the phone can be paused such that the user will not incur additional battery drain. The primary difference is the user is provided the option to use additional battery power to also playback media on their phone, if they so desire.

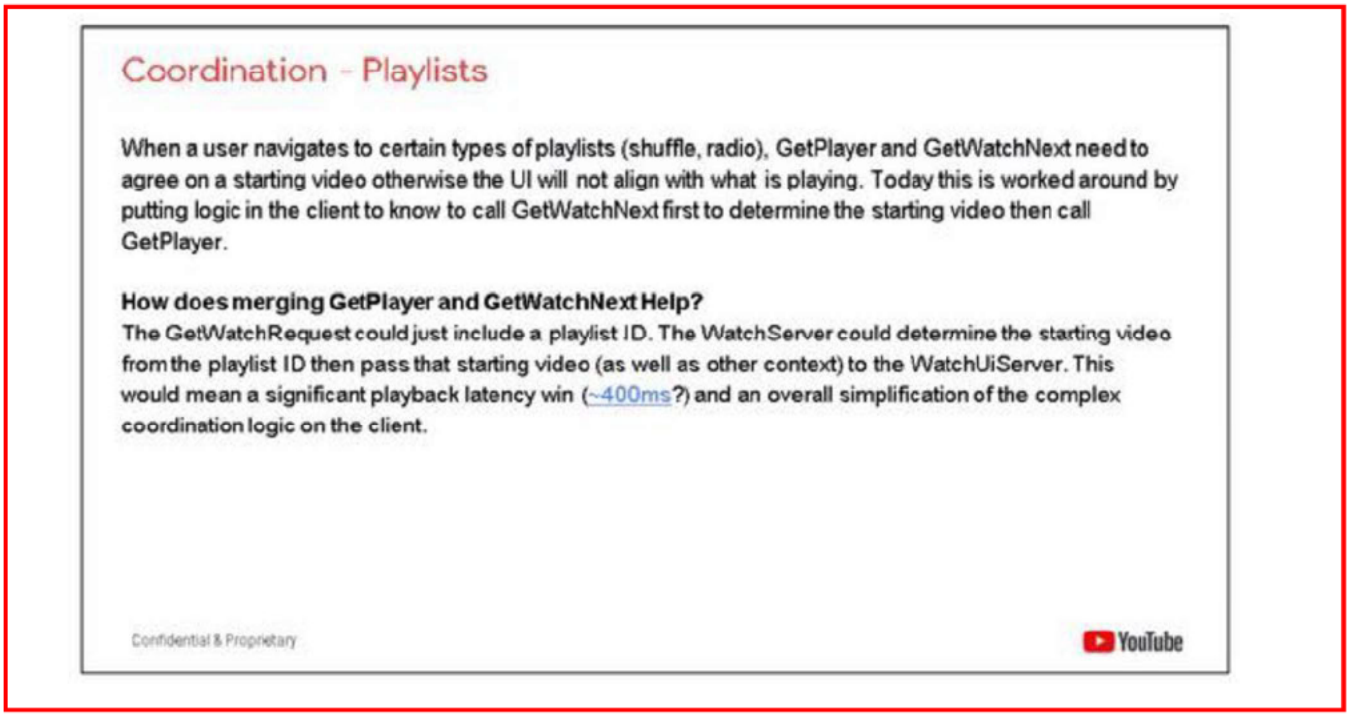
B. Non-Infringing Alternative - Cloud Service, Not Playback Device, Communicates With Cloud Servers To Identify the Next One Or More Media Items In The Remote Playback Queue

287. All of the asserted claims an "instruction configures the at least one given playback device to (i) communicate with the cloud-based computing system in order to obtain data

identifying a next one or more media items that are in the remote playback queue, (ii) use the obtained data to retrieve at least one media item in the remote playback queue from the cloud-based media service; and (iii) play back the retrieved at least one media item." Dr. Schmidt opines that after receiving a setPlaylist message (the alleged "instruction") a receiver device (the alleged "playback device") is "configured" to "obtain data identifying a next one or more media items that are in the remote playback queue" by transmitting a GetWatchNext request to the servers of the GetWatchNext Service, and that the receiver device uses the obtained data to "retrieve at least one media item in the remote playback queue from the cloud-based media service" by sending a getPlayer request to the servers of the Player Service to retrieve the media content for playback. Schmidt Rpt., ¶¶162-166, 323-324. For the reasons I discussed above, I disagree with Dr. Schmidt that the accused YouTube system satisfies this limitation of the asserted claims.

288. Even if Dr. Schmidt's infringement opinions were accepted, as I showed in my opening report, a non-infringing alternative that was available to Google at the time the '033 patent issued was for the receiver device *not* to send the accused getWatchNext and/or getPlayer requests. Opening '033 Rpt., ¶¶757-762. Instead, the receiver device would send a request to a Onesie agent. *Id*. The Onesie agent would then make the getWatchNext and/or getPlayer calls (which could be separate or combined) and stream back the media content to the receiver device for the videos in the playlist. *Id*.. Thus, in this alternative the receiver device does not "(i) communicate with the cloud-based computing system in order to obtain data identifying a next one or more media items that are in the remote playback queue," and does not "(ii) use the obtained data to retrieve at least one media item in the remote playback queue from the cloud-based media service," even under Dr. Schmidt's interpretation of the claims.

289. In response, Dr. Schmidt begins by stating that "Google has not provided sufficient details as to the contents of the 'request' sent from the Receiver to a 'Onesie agent,'" and that he therefore "do[es] not have enough information to fully evaluate whether this alleged alternative would have been non-infringing, available, technically feasible, or commercially acceptable." Schmidt Rpt., ¶498. But Dr. Schmidt does not identify the additional details he requires in order to "fully evaluate" this non-infringing alternative. For example, with the now launched Onesie feature the request from the receiver to the Onesie agent would include the videoId and/or playlist ID for the Shared Queue. As another example, in the Streaming Watch alternative I discuss below the request to the Onesie agent would include a playlist ID and any associated servers could then determine the starting video from the playlist ID and stream the content for the starting video and subsequent videos in the playlist to the receiver:



See GOOG-SONOSNDCA-00070863 (Streaming Watch (Everywhere?)) at -913 ("GetWatchRequest could just include a playlist ID. The WatchServer could determine the

starting video from the playlist ID then pass that starting video (as well as other context) to the WatchUiServer.").

290. In support of my opinion that this non-infringing alternative was technically feasible, I showed in my Opening '033 Report that Google had implemented a feature called Onesie and was working on a further feature called Streaming Watch. Opening '033 Report, ¶760. In response, Dr. Schmidt states that he has seen evidence that "undermines Google's assertion that 'Google has implemented Onesie for many receiver devices and is currently working on a feature called Streaming Watch.'" Schmidt Rpt., ¶499.

291. More specifically, Dr. Schmidt points to testimony from Vincent Mo, who he refers to as "Google's 30(b)(6) witness for YouTube," and states that Mr. Mo was "unaware of any plans to implement changes required for Google's alleged alternatives." *Id.* Initially, I understand that Mr. Mo was Google's 30(b)(6) witness for only certain topics in Sonos's 30(b)(6) notice that related to the functionality that is accused of infringement in this case—I understand he was not designated on any topics directed at non-infringing alternatives for YouTube. *See* Mo Tr. at 17:2-6 ("We have designated Mr. Mo on 3 Topic Number 1, subparts romanettes 2 through 7 as those related to the accused casting functionality in YouTube, YouTube Music, YouTube Kids and YouTube TV"); Mo Tr., Ex. 2 (Sonos's 30(b)(6) Notice). Moreover, in the testimony Dr. Schmidt cites Mr. Mo confirms that he was aware of Google's work on Onesie. Mo Tr. at 223:20-22 ("Q. My understanding is that it [Onesie] has something to do with streaming content. A. I have heard of onesie."). Unlike the simplistic system of the '033 patent, "YouTube has lots of systems." Mo Tr. at 224:4-14. Mr. Mo testified the work on Onesie and Streaming Watch are performed by a different team of Google engineers and is "abstracted out from" the work that he does. Mo Tr. at 224:4-14, 140:14-18. Thus, it is not surprising that not every engineer is aware of every system.

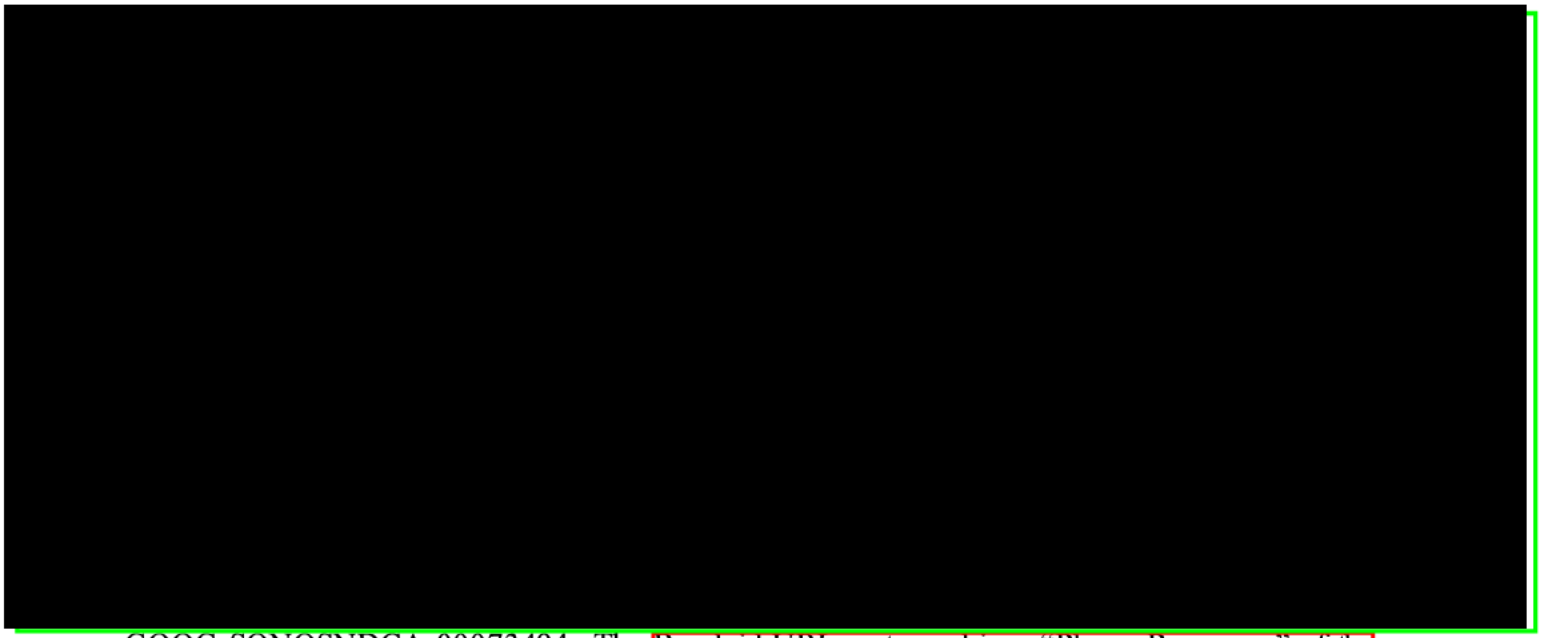
videoIds to the shared queue, RQ playlist…I mentioned earlier that MDx, these playlists are all prefixed with the two letter RQ. That standards for remote queue."); GOOG-SONOSWDTX-00052992 (MDx Communication Protocol v3 Differences) at -3000 (listId is "prefixed with RQ" when playing from "the remote queue."). As a result, the scope of this non-infringing alternative could be narrowly targeted to playlistIds of type RQ (not all playlists) and still avoid infringement.

294. Dr. Schmidt goes on to say that "it is unclear to me that this alleged alternative would not still literally infringe or at least infringe under the Doctrine of Equivalents (DoE)." Schmidt Rpt., ¶501. Initially, Dr. Schmidt fails to provide any opinion under DoE. As to literal infringement, Dr. Schmidt provides only a conclusory infringement analysis. I have made my best effort to respond to Dr. Schmidt's conclusory assertion, but to the extent Dr. Schmidt is permitted to provide additional details or explanation I reserve the right to respond.

295. Dr. Schmidt provides a conclusory assertion that a receiver device with just the "Onesie" feature would still infringe by communicating with a Onesie agent to "obtain data (e.g., Bandaid URL) identifying a next one or more media items that are in the remote playback queue and the Receiver would still use the obtained data (e.g., Bandaid URL) to retrieve at least one media item in the remote playback queue (e.g., certain 'chunks of media content')." Schmidt Rpt., ¶501. Dr. Schmidt provides no further analysis supporting his assertion that Onesie would infringe the asserted claims. Dr. Schmidt's opinion is flawed and I disagree that a playback device that implements Onesie would infringe.

296. A client device (e.g., a receiver device) implementing Onesie would send a Onesie request for a media item (e.g., a song or video) each time it starts playback of the next media item. The Onesie response received by the client is shown below. As can be seen, the Onesie response returns media content for the beginning of the song or video (the blocks labeled "V" and "A")

followed by the "Player Response" containing the Bandaid URLs that the receiver device can use to request the remaining chunks of media content:



GOOG-SONOSNDCA-00073494. The Bandaid URLs returned in a "Player Response" of the Onesie request are *not* "data identifying a next one or more media items that are in the [alleged] remote playback queue" or data that a receiver device uses to "retrieve at least one media item in the [alleged] remote playback queue." The Bandaid URLs in the Player Response do not correspond to a media item. Rather, as Dr. Schmidt concedes, the Bandaid URLs in a Onesie response can, at most, correspond to "certain 'chunks of media content'" for a media item, namely those chunks that come after the portion of the media item provided in the Onesie response. *See* Schmidt Rpt., ¶501. In the accused YouTube system the "media items" selected for playback are complete songs or videos, as Dr. Schmidt himself acknowledges. *See* Schmidt Rpt., ¶501 (stating that YouTube allows "a user to select a single media item (e.g., a song, video, or on-demand TV program) for playback"). Thus, I disagree with Dr. Schmidt that the Bandaid URLs in a Onesie response can be used to identify or retrieve a media item.

297. Further, although Dr. Schmidt accuses a videoId of being "data identifying a next one or more media items that are in the remote playback queue" and data used to "retrieve at least one media item in the remote playback queue" in his infringement analysis, Dr. Schmidt does not provide any opinion that the Onesie implementation would infringe based on a videoId. *See* Schmidt Rpt., ¶501. A videoId is not data that is used to "retrieve at least one media item in" a queue. At best, a videoId is used to retrieve the first several seconds of media content. *See also, e.g.*, GOOG-SONOSNDCA-00073494, 95 ("Onesie only streams the first seven seconds of a video. For the rest of the video, the Player needs to fetch it from the content-hosting machine."). A person of skill in the art would understand that retrieving the first several seconds of a media item is not the same as retrieving the media item. Indeed, in the YouTube system a "videoId" is an identifier for a complete song or video. It is not an identifier for the beginning portion of a song or video. Thus, to the extent Dr. Schmidt alleges a videoId is the claimed "data identifying" a "media item," that videoId must be used to retrieve the complete media item identified by the videoId, not just a few seconds of the media item.

298. While Onesie as implemented does not infringe, I also opined that Google could extend Onesie further so that "[t]he Onesie agent would send the GetWatchNext request and GetPlayer request, and then stream the media data from the Onesie agent to the receiver device" for the videos in the playlist. Opening '033 Report, ¶759. In this regard, Google is already developing a feature called "Streaming Watch" in which a playback device could send a request to a Onesie agent with just a playlistId and the Onesie agent would then make the necessary WatchNext and GetPlayer calls (which could be merged) to identify a starting video and stream back media content. Dr. Schmidt alleges that this non-infringing alternative would still infringe because, according to Dr. Schmidt, a receiver device would use "WatchNextResponse data" to

retrieve a media item. Schmidt Rpt., ¶503. But Dr. Schmidt misunderstands this alternative. Google's documents indicate that it has discussed different options for implementing Streaming Watch. For playing back playlists on a receiver device, Streaming Watch could be implemented such that the Onesie agent (and not the receiver) would use the playlistId to identify the starting and subsequent media items in the playlist and stream back the media content to the receiver device. *See* GOOG-SONOSNDCA-00070863 (Streaming Watch (Everywhere?)) at -913 ("GetWatchRequest could just include a playlist ID. The WatchServer could determine the starting video from the playlist ID then pass that starting video (as well as other context) to the WatchUiServer."). A receiver device would not use a videoId ("WatchNextResponse data") to retrieve the content for the media items in the receiver device.

299. Dr. Schmidt also argues that this non-infringing alternative would not be technically feasible because Google's "internal documents on 'Onesie' highlight security risks if Google were to consolidate its servers." Schmidt Rpt., ¶507 (citing GOOG-SONOS-NDCA-00086299). But the document Dr. Schmidt cites to merely discusses "security considerations" and the manner in which Onesie would address them. GOOG-SONOS-NDCA-00086299 at -307. The fact that Google analyzed security considerations for a new feature does not show that the feature was not "technically feasible." Rather, it shows good engineering practice.

300. Dr. Schmidt also opines that "'Onesie' is, at most, only applicable to a Sender device." Schmidt Rpt., ¶507. But I understand that Google has now launched Onesie for Cast receiver devices.[21] *See also* GOOG-SONOSNDCA-00116349; GOOG-SONOSNDCA-00116350. That Onesie has now been launched for Cast receiver devices confirms that Onesie is technically feasible.

[21] Conversation with Pawel Jurzyk

I, Samrat Bhattacharjee, declare under penalty of perjury under the laws of the United States that the foregoing is true and correct.

DATED: January 13, 2023

Samrat Bhattacharjee